INForum 2017

Atas do Nono Simpósio de Informática

12 e 13 de outubro de 2017

Universidade de Aveiro

Aveiro, Portugal

Ficha Técnica

Título:	INFORUM 2017 - Atas do Nono Simpósio de Informática
Coordenação:	João Paulo Barraca, Helena Rodrigues, António Teixeira, José Maria Fernandes
ISBN:	978-972-789-522-9
Ano:	2017
Design e Paginação:	João Paulo Barraca
Editora:	UA Editora
	Universidade de Aveiro
	Serviços de Biblioteca, Informação Documental e Museologia
	1.a edição outubro 2017

publicado em formato eletrónico nos seguinte endereços web:
http://inforum.org.pt/INForum2017/docs/atas-do-inforum2017
http://ria.ua.pt/handle/10773/18582

Organização

Apoios

Prefácio

Este volume contém as atas da 9.a edição do Simpósio em Informática, INForum 2017, a qual decorreu no Pavilhão de Exposições de Aveiro, em Aveiro, conjuntamente com o TechDays 2017, nos dias 12 e 13 de outubro de 2017.

O INForum é um evento privilegiado de reunião da comunidade nacional nas diversas vertentes da informática, bem como um fórum de eleição para a divulgação, discussão e reconhecimento de trabalhos científicos e avanços tecnológicos em informática. O INForum orgulha-se de ser um palco especializado para promover, por um lado, o intercâmbio de conhecimento e experiência entre a academia e a indústria e, por outro lado, a estreia de jovens investigadores que procuram a divulgação, a crítica construtiva e o encorajamento ao seu trabalho.

Esta edição do INForum contou com um conjunto de dez sessões em áreas relevantes e atuais da informática, de comprovado interesse e dinamismo no espaço académico e industrial nacional:

- Bioinformática (BIO)
- Ciência e Engenharia de Software (SOFT-PT)
- Computação Móvel e Ubíqua (CMU)
- Computação Paralela, Distribuída e de Larga Escala (CPDLA)
- Comunicações e Redes de Computadores (CRC)
- Gestão de Dados e Conhecimento (GDC)
- Human Centred Pervasive Systems (HuPS)
- Sistemas de Informação Empresariais (BusIS)
- Sistemas Embebidos e de Tempo Real (SETR)
- Segurança de Sistemas de Computadores e Comunicações (SSCC)

Nesta edição foram aceites dois tipos de contribuições: *artigos*, apresentando resultados de trabalho de investigação científica realizado em contexto académico ou industrial;*comunicações*, cujo objetivo é permitir a divulgação de uma forma mais expedita e informal de trabalho de I&D desenvolvido na comunidade, através de uma comunicação oral curta e/ou da exposição de um poster. No conjunto das sessões, foram submetidos 79 trabalhos, 58 artigos e 21 comunicações, os quais foram sujeitos a um processo de revisão por pares pelas respectivas comissões de programa. Cada artigo submetido recebeu pelo menos 3 revisões e as comunicações receberam pelo menos 2 revisões.

As comissões de programa das várias sessões temáticas discutiram os trabalhos submetidos, tendo selecionado para inclusão no programa técnico 30 artigos completos, que estão incluídos nestas atas organizados por sessão, e 33 comunicações (incluindo os trabalhos submetidos como artigo, mas convertidos em comunicações). A todos agradecemos o esforço e cuidado colocado na análise e discussão dos artigos submetidos, e a contribuição para um programa técnico diversificado e abrangente.

Para além dos artigos e comunicações aceites para apresentação, este ano o programa técnico foi complementado com duas palestras convidadas dadas por Eleni Pratsini, Diretora da divisão Cognitive IoT Solutions na IBM Research, em Zurique e Liliana Ferreira, Diretora do Fraunhofer Institute for Assistive Information and Communication Solutions (AICOS) em Portugal e Professora Convidada na Faculdade de Engenharia da Universidade do Porto. Os resumos de ambas as palestras estão incluídos nestas atas.

Este ano, o INForum 2017 decorreu enquadrado no evento Techdays 2017, partilhando a localização e algumas sessões. Esta foi uma oportunidade única que não podia deixar de ser aproveitada. A organização do Techdays 2017 estimou que o evento fosse visitado por 10000 pessoas, entre alunos, académicos,

investigadores e público em geral. Também estiveram presentes dezenas de empresas, algumas com forte impacto de inovação e com uma longa cooperação com as universidades e institutos representados pelos artigos do INForum. Esta oportunidade constitui uma mais valia estratégica para a aproximação da academia e indústria, objetivos basilares do INForum.

O nosso trabalho de Presidentes da Comissão de Programa do INForum 2017 apenas foi possível devido à preciosa colaboração que recebemos ao longo deste ano de diversos colegas e entidades, sendo que, pela sua colaboração, agradecemos de forma particular.

À Comissão Coordenadora do INForum, e muito especialmente ao Miguel Correia, agradecemos a confiança depositada em nós e o apoio e orientação fornecida sempre que precisamos.

Aos coordenadores das várias sessões e respetivas comissões de programa, agradecemos todo o esforço colocado na divulgação das sessões e no processo de revisão e avaliação dos trabalhos submetidos, por vezes num período muito curto.

Às oradoras convidadas, Eleni Pratsini e Liliana Ferreira, agradecemos terem aceitado o convite que permitiu enriquecer o programa do INForum com a sua presença e as suas palestras.

Aos membros do júri de atribuição do prémio de melhor artigo para estudante, Miguel Rio, David Lamas e Paulo Cortez, o nosso especial agradecimento por terem aceitado o nosso convite e pela colaboração prestada.

À Associação *Portuguese Chapter da Internet Society* (ISOC-PT), o nosso agradecimento por instituir este ano o prémio ISOC-PT, cumprindo assim os seus objectivos de *promover uma Internet aberta, inclusiva e acessível a todos, confiável e que seja um factor de desenvolvimento económico, humano e social de toda a humanidade*. O nosso agradecimento também ao júri de atribuição deste prémio, presidido por José Legatheaux Martins.

A realização do INForum 2017 apenas foi possível graças ao empenho e dedicação de muitas outras pessoas e entidades, às quais agradecemos:

À Comissão Organizadora agradecemos todo o auxílio prestado e o excelente trabalho de organização local. Ao José Luís Faria agradecemos a disponibilidade e a colaboração no papel de webmaster. À Universidade de Aveiro, ao Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA) e ao Instituto de Telecomunicações (IT), estamos reconhecidos pela disponibilidade para apoiar o evento e pelos recursos fornecidos. Agradecemos ainda aos nossos patrocinadores, parceiros e apoiantes do INForum. Um agradecimento especial para a organização do TechDays 2017 e para a Câmara de Aveiro, Internet Society (ISOC), e às empresas Maxdata, Outsystems, IBM, e FCA.

Por último, gostaríamos de agradecer aos autores dos artigos e comunicações submetidas ao INForum 2017 pelo seu interesse no evento, sem o qual o evento não seria possível. Esperamos contar convosco de novo no INForum 2018!

Aveiro e Guimarães, outubro de 2017

António Teixeira e Helena Rodrigues

Comité Técnico

Presidência da Comissão de Programa

Helena Rodrigues ..... Universidade do Minho
António Teixeira ..... Universidade de Aveiro

Comissão Organizadora

António Teixeira ..... IEETA, DETI - Universidade de Aveiro (Presidente)
José Maria Fernandes ..... IEETA, DETI - Universidade de Aveiro
João Paulo Barraca ..... IT, DETI - Universidade de Aveiro
José Luís Faria ..... Universidade do Minho (webmaster)
Samuel Silva ..... IEETA, DETI - Universidade de Aveiro
Nuno Almeida ..... IEETA, DETI - Universidade de Aveiro

Comissão Coordenadora

Miguel P. Correia ..... Universidade de Lisboa (Presidente)
André Zúquete ..... Universidade de Aveiro
Antónia Lopes ..... Universidade de Lisboa
António Teixeira ..... Universidade de Aveiro
Beatriz Sousa Santos ..... Universidade de Aveiro
Helena Rodrigues ..... Universidade do Minho
João Coelho Garcia ..... Universidade de Lisboa
João Pascoal Faria ..... Universidade do Porto
José Orlando Pereira ..... Universidade do Minho
Lúcio Ferrão ..... OutSystems
Luís Veiga ..... Universidade de Lisboa
Nuno Preguiça ..... Universidade Nova de Lisboa
Ricardo Rocha ..... Universidade do Porto
Salvador Abreu ..... Universidade de Évora
Simão Melo de Sousa ..... Universidade da Beira Interior

Comissões de Programa das Sessões Temáticas

Bioinformática

Coordenação:
José Luís Oliveira ..... DETI - Universidade de Aveiro
Sérgio Matos ..... DETI - Universidade de Aveiro

Alexandra Carvalho ..... IT, Instituto Superior Técnico - Universidade de Lisboa
Alexandre P. Francisco ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Ana T. Freitas ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
André Falcão ..... LASIGE, Faculdade de Ciências - Universidade de Lisboa
Arlindo Oliveira ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Armando Pinho ..... DETI - Universidade de Aveiro
Claudine Chaouiya ..... Instituto Gulbenkian de Ciência
Francisco Couto ..... LASIGE, Faculdade de Ciências - Universidade de Lisboa
João A. Carriço ..... IMM, Faculdade de Medicina - Universidade de Lisboa

Joel Arrais ..... DEI - Universidade de Coimbra
Ludwig Krippahl ..... CENTRIA, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa
Mário Silva ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Matthias E. Futschik ..... CBME, CCMAR, Universidade do Algarve
Miguel Rocha ..... CCTC, Universidade do Minho
Pedro T. Monteiro ..... INESC-ID
Rui Camacho ..... Faculdade de Engenharia - Universidade do Porto
Rui Mendes ..... CCTC, Universidade do Minho
Sara C. Madeira ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Susana Vinga ..... IDMEC, Instituto Superior Técnico - Universidade de Lisboa
Vítor Costa ..... Faculdade de Ciências - Universidade do Porto

Ciência e Engenharia de Software

Coordenação:
João Costa Seco ..... Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa

Ademar Aguiar ..... Faculdade de Ciências - Universidade do Porto
Ana Matos ..... Instituto Superior Técnico - Universidade de Lisboa
André Santos ..... ISCTE - Instituto Universitário de Lisboa
António Rito Silva ..... Instituto Superior Técnico - Universidade de Lisboa
Bruno Cabral ..... Universidade de Coimbra
Carla Ferreira ..... Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa
Daniela da Cruz ..... Checkmarx
Francisco Martins ..... Faculdade de Ciências - Universidade de Lisboa
Hugo Lourenço ..... OutSystems
Hugo Vieira ..... IMT Lucca
Isabel Sofia Brito ..... Instituto Politécnico de Beja
Jorge Sousa Pinto ..... Universidade do Minho
José Luís Oliveira ..... Universidade de Aveiro
João Pascoal de Faria ..... Faculdade de Ciências - Universidade do Porto
João Paulo Fernandes ..... Universidade de Coimbra
João Saraiva ..... Universidade do Minho
Luís Lopes ..... Faculdade de Ciências - Universidade do Porto
Luís Soares Barbosa ..... Universidade do Minho
Marco Vieira ..... Universidade de Coimbra
Miguel Goulão ..... Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa
Mário Florido ..... Faculdade de Ciências - Universidade do Porto
Nuno Silva ..... Critical Software
Rui Maranhão ..... Instituto Superior Técnico - Universidade de Lisboa
Salvador Pinto Abreu ..... Universidade de Évora
Simão Melo de Sousa ..... Universidade da Beira Interior
Vasco Thudichum Vasconcelos ..... Faculdade de Ciências - Universidade de Lisboa
Vítor Santos Costa ..... Faculdade de Ciências - Universidade do Porto

Computação Móvel e Ubíqua

Coordenação:
Susana Sargento ..... Universidade de Aveiro
Nuno Cruz ..... Instituto Politécnico de Lisboa

Adriano Moreira ..... Universidade do Minho
Ana Paula Afonso ..... Universidade de Lisboa

António Coelho ..... Universidade do Porto
Carlos Baquero ..... Universidade do Minho
Dulce Domingues ..... Universidade de Lisboa
Filipe Pacheco ..... Instituto Superior de Engenharia do Porto
Frutuoso Silva ..... Universidade da Beira Interior
Hugo Miranda ..... Universidade de Lisboa
João Barreto ..... Universidade de Lisboa
Jorge Sá Silva ..... Universidade de Coimbra
Paulo Ferreira ..... Universidade de Lisboa
Pedro Araújo ..... Universidade da Beira Interior
Pedro Brandão ..... Universidade do Porto
Rui José ..... Universidade do Minho
Rui Prior ..... Universidade do Porto
Sérgio Duarte ..... Universidade Nova de Lisboa
Teresa Romão ..... Universidade Nova de Lisboa

Computação Paralela, Distribuída e de Larga Escala

Coordenação:
Paula Prata ..... Universidade da Beira Interior
Miguel Matos ..... Universidade de Lisboa

Alysson Bessani ..... Universidade de Lisboa
Lúcio Ferrão ..... OutSystems
Luís Rodrigues ..... Universidade de Lisboa
José Simão ..... Instituto Superior de Engenharia de Lisboa
João Leitão ..... Universidade Nova de Lisboa
João Lourenço ..... Universidade Nova de Lisboa
João Nuno Silva ..... Universidade de Lisboa
Hugo Miranda ..... Universidade de Lisboa
Hervé Paulino ..... Universidade Nova de Lisboa
Francisco Maia ..... Universidade do Minho
Nuno Laranjeiro ..... Universidade de Coimbra
Ricardo Dias ..... Suse
Ricardo Vilaça ..... Universidade do Minho
Rómulo Gonçalves ..... Netherlands eScience Center (NLeSC)
Salvador Abreu ..... Universidade de Évora
Óscar Mortágua Pereira ..... Universidade de Aveiro
João Barreto ..... Universidade de Lisboa

Comunicações e Redes de Computadores

Coordenação:
Marília Curado ..... Universidade de Coimbra
Daniel Corujo ..... Universidade de Aveiro
João Vilela ..... Universidade de Coimbra

Adriano Moreira ..... Universidade do Minho
Amaro Sousa ..... Universidade de Aveiro
Augusto Casaca ..... INESC-ID
Bruno Sousa ..... Universidade de Coimbra
Carlos Rabadão ..... Instituto Politécnico de Leiria
Carlos Serôdio ..... Universidade de Trás-os-Montes e Alto Douro

Edmundo Monteiro ..... Universidade de Coimbra
Fernando Boavida ..... Universidade de Coimbra
Joaquim Ferreira ..... Universidade de Aveiro
Joaquim Macedo ..... Universidade do Minho
Jorge Sá Silva ..... Universidade de Coimbra
José Legatheaux Martins ..... Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa
Luís Lino Ferreira ..... Instituto Superior de Engenharia do Porto
Manuel Ricardo ..... Faculdade de Ciências - Universidade do Porto
Maria João Nicolau ..... Universidade do Minho
Noelia Correia ..... Universidade do Algarve
Paulo Pedreiras ..... Universidade de Aveiro
Paulo Rogério Barreiros dAlmeida Pereira ..... Instituto de Engenharia de Sistemas e Computadores
Paulo Martins de Carvalho ..... Universidade do Minho
Paulo Simões ..... Universidade de Coimbra
Pedro Salgueiro ..... Universidade de Évora
Pedro Sousa ..... Universidade do Minho
Rui Aguiar ..... Universidade de Aveiro
Solange Rito Lima ..... Universidade do Minho
Vasco Pedro, ... Universidade de Évora
Vasco Pereira ..... Universidade de Coimbra

Gestão de Dados e Conhecimento

Coordenação:
Helena Galhardas ..... Instituto Superior Técnico - Universidade de Lisboa
Nuno Bettencourt ..... DEI - Instituto Politécnico do Porto
Rui Camacho ..... Faculdade de Engenharia - Universidade do Porto

Alberto Simões ..... Universidade do Minho
Bruno Martins ..... Instituto Superior Técnico - Universidade de Lisboa
Carlos Ferreira ..... DEI - Instituto Politécnico do Porto
Carlos Soares ..... Faculdade de Engenharia - Universidade do Porto
Cristina Ribeiro ..... Faculdade de Engenharia - Universidade do Porto
David Matos ..... Instituto Superior Técnico - Universidade de Lisboa
Diana Santos ..... University of Oslo
Francisco Couto ..... Universidade de Lisboa
Fernando Batista ..... ISCTE - Instituto Universitário de Lisboa
Irene Rodrigues ..... Universidade de Évora
João Dias Pereira ..... Instituto Superior Técnico - Universidade de Lisboa
João Paulo Cordeiro ..... Universidade da Beira Interior
Jorge Barbosa ..... Faculdade de Engenharia - Universidade do Porto
Luis Teixeira ..... Faculdade de Engenharia - Universidade do Porto
Nuno Mamede ..... Instituto Superior Técnico - Universidade de Lisboa
Nuno Silva ..... DEI - Instituto Politécnico do Porto
Paulo Maio ..... DEI - Instituto Politécnico do Porto
Paulo Novais ..... Universidade do Minho
Paulo Oliveira ..... DEI - Instituto Politécnico do Porto
Paulo Quaresma ..... Universidade de Évora
Pável Calado ..... Instituto Superior Técnico - Universidade de Lisboa
Pedro Ferreira ..... Instituto de Investigação e Inovação em Saúde, Universidade do Porto
Ricardo Ribeiro ..... ISCTE - Instituto Universitário de Lisboa

Human Centred Pervasive Systems

Coordenação:
José Maria Fernandes ..... IEETA, DETI - Universidade de Aveiro

Ana Fred ..... IT, Instituto Superior Técnico - Universidade de Lisboa
Carlos Teixeira ..... Universidade de Lisboa
Cesar Analide ..... Universidade do Minho
Helder Oliveira ..... INESC-TEC, Universidade do Porto
Hugo Paredes ..... Universidade de Trás-os-Montes e Alto Douro
Hugo Plácido da Silva ..... Instituto de Telecomunicações
Ilídio C. Oliveira ..... IEETA, DETI - Universidade de Aveiro
João Dias ..... IT, Instituto Superior Técnico - Universidade de Lisboa
João Sanches ..... Instituto Superior Técnico - Universidade de Lisboa
Miguel Coimbra ..... Universidade do Porto
Paulo Carvalho ..... Universidade de Coimbra
Paulo Novais ..... Universidade do Minho
Raquel Sebastião ..... Universidade de Aveiro
Sergi Bermudéz ..... Madeira-ITI, Universidade da Madeira
Susana Brás ..... Universidade de Aveiro
Vitor Santos ..... Universidade Nova de Lisboa

Segurança de Sistemas de Computadores e Comunicações

Coordenação:
Filipe Araújo ..... Universidade de Coimbra
Nuno Santos ..... Instituto Superior Técnico - Universidade de Lisboa

André Zúquete ..... IEETA, Universidade de Aveiro
Carlos Serrão ..... ISTAR-IUL, ISCTE - Instituto Universitário de Lisboa
Dulce Domingos ..... LASIGE, Faculdade de Ciências Universidade de Lisboa
Edmundo Monteiro ..... Universidade de Coimbra
Henrique Domingos ..... Universidade Nova de Lisboa
Henrique Santos ..... Universidade do Minho
Ibéria Medeiros ..... LASIGE, Faculdade de Ciências - Universidade de Lisboa
José Alegria ..... PT - Portugal Telecom
José Carlos Lourenço Martins ..... Academia Militar
José Manuel Valença ..... Universidade do Minho
Luís Antunes ..... IT-SQIG, DCC - Universidade do Porto
Manuel Barbosa ..... Universidade do Porto
Marco Vieira ..... Universidade de Coimbra
Mário Zenha Rela ..... Universidade de Coimbra
Miguel Pardal ..... Instituto Superior Técnico - Universidade de Lisboa
Miguel Pupo Correia ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Nuno Neves ..... LASIGE, Faculdade de Ciências Universidade de Lisboa
Patrício Rodrigues Domingues ..... Instituto Politécnico de Leiria
Paulo Sousa ..... Maxdata Software
Pedro Adão ..... Instituto Superior Técnico - Universidade de Lisboa
Pedro Inácio ..... Universidade da Beira Interior
Ricardo Chaves ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Rui Miguel Silva ..... UbiNET, Instituto Politécnico de Beja
Salvador Abreu ..... Universidade de Évora
Sandro Pinto ..... Universidade do Minho

Sistemas de Informação Empresariais

Coordenação:
Sérgio Guerreiro ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa

Ana Alice Baptista ..... ALGORITMI, Universidade do Minho
Ana Respicio ..... Faculdade de Ciências - Universidade de Lisboa
André Vasconcelos ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
António Gonçalves ..... Instituto Politécnico de Setúbal
Carlos Teixeira ..... Faculdade de Ciências - Universidade de Lisboa
Duarte Gouveia ..... Universidade da Madeira
Elsa Cardoso ..... INESC-ID, ISCTE - Instituto Universitário de Lisboa
João Barata ..... Universidade Coimbra
José Barateiro ..... Laboratório Nacional de Engenharia Civil
José Borbinha ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
José Ramalho ..... Universidade do Minho
José Tribolet ..... INESC-ID, Instituto Superior Técnico - Universidade de Lisboa
Miguel Ferreira ..... KEEP SOLUTIONS
Miguel Mira da Silva ..... INOV, Instituto Superior Técnico - Universidade de Lisboa
Nuno Pombo ..... Universidade da Beira Interior
Rosário Bernardo ..... Instituto Superior Técnico - Universidade de Lisboa
Rui Marques ..... Universidade de Aveiro

Revisores Adicionais

Mário Rodrigues ..... IEETA, ESTGA - Universidade de Aveiro
Liliana Ferreira ..... AICOS - Fraunhofer Portugal
Eduardo Pinho ..... IEETA - Universidade de Aveiro
Pedro Sernadela ..... IEETA - Universidade de Aveiro
André Lamúrias ..... LASIGE, Faculdade de Ciências - Universidade de Lisboa
Cátia Vaz ..... Instituto Politécnico de Lisboa
Hélder Gomes ..... IEETA, ESTGA - Universidade de Aveiro
Diogo Lima ..... LASIGE, Faculdade de Ciências Universidade de Lisboa
Jan Cederquist ..... IT, Instituto Superior Técnico - Universidade de Lisboa
Samuel Neves ..... IEETA, DETI - Universidade de Aveiro
Jaime Correia ..... DEI - Universidade de Coimbra

Índice de Artigos

Dia 1, 12 de outubro

Human Centred Pervasive Systems

Bioinformática

Comunicações e Redes de Computadores

Sistemas de Informação Empresariais

Dia 2, 13 de outubro

Sistemas Embebidos e de Tempo Real

Gestão de Dados e Conhecimento

Ciência e Engenharia de Software

Computação Móvel e Ubíqua

Segurança de Sistemas de Computadores e Comunicações

Computação Paralela, Distribuída e de Larga Escala

Índice de Autores

Sessões Convidadas

Dr. Eleni Pratsini - IBM Research Zurich

Industry 4.0 & Cognitive IoT: Challenges and Opportunities

Resumo

Todays high-tech firms are challenged to continuously transform their business models and partner ecosystems to keep pace with the quickly evolving nature of business technology. Across the industry, companies are turning their focus from traditional business equipment to a new generation of devices that will transform not just the electronics industry but many others. This presentation will examine how companies are moving beyond merely selling connected, intelligent products and services to using cognitive IoT to deliver compelling customer experiences over the life of their products.

Biografia

Eleni Pratsini é Diretora da divisão Cognitive IoT Solutions na IBM Research, nos laboratórios de Zurique. Dirige as atividades da divisão relacionadas com soluções para a Internet das Coisas, explorando Inteligência Artificial e Computação Cognitiva.

No período 2013-2016, a Doutora Eleni Pratsini foi Diretora do IBM Research Lab na Irlanda, coordenando as atividades do laboratório na área de Smarter Cities and Cognitive IoT, produzindo avanços científicos e tecnológicos para sistemas urbanos e ambientais inteligentes, com um foco na criação e optimição de soluções e sistemas energeticamente sustentáveis, com recursos limitados (ex, sistemas de gestão das redes de distribuição de água), transportes, e o restante tecido tecnológico que assimila e partilha dados e modelos para estes domínios.

Previamente, a Doutora Eleni Pratsini foi ainda Diretora da divisão Optimization Research no IBM T.J. Watson Research Center em Nova York, Coordenadora das áreas de Mathematical and Computational Sciences na IBM Research - Zurique, e Investigadora no Swiss Federal Institute of Technology Zurich (ETHZ).

Eleni Pratsini possui um Doutoramento em Análise Quantitativa e Engenharia Industrial (Universidade de Cincinnati, EUA), um MBA em Finanças (UCLA, EUA) e uma Licenciatura em Engenharia Civil (Universidade de Birmingham, UK).

Liliana Ferreira - Fraunhofer AICOS

Information in healthcare: a healthier future

Resumo

Understanding the users, their activities, context and behavior and providing tailored recommendations to patients and health professionals are some of the potential applications of automatic tools that extract information from structured and unstructured data sources and perform accurate automatic mapping of free data onto a structured knowledge representation.

This talk provides a short walk through the opportunities and challenges of the application of information and communications technologies to the delivery of safe, effective, timely, patient-centered, and efficient health care.

Biografia

Liliana Ferreira é Diretora do Fraunhofer Institute for Assistive Information and Communication Solutions (AICOS) em Portugal e Professora Convidada na Faculdade de Engenharia - Universidade do Porto.

Estudou Matemática Aplicada na Faculdade de Ciências - Universidade do Porto, e Engenharia Electrónica e de Telecomunicações na Universidade de Aveiro. Os seus principais interesses de investigação centram-se nas Human Language Technologies, Medical Knowledge Representation, Information Extraction e Health Informatics. Possui um longo percurso de investigação, ligado à indústria e instituições de investigação como a Philips Research em Eindhoven, a IBM Research & Development em Böblingen, o Instituto de Engenharia Electrónica e Telemática de Aveiro, o Ubiquitous Knowledge Lab na Universidade Técnica de Darmstadt e na Universidade de Tübingen.

Liliana Ferreira possui um Doutoramento em Engenharia Informática (Universidade de Aveiro).

Second Language Learning Using Bilingual eBooks

Tiago Soares[1], Xavier Anguera[2], and Isabel Trancoso[3]

[1] Instituto Superior Técnico / INESC-ID tvdsoares@gmail.com
[2] ELSA Corp. xanguera@gmail.com
[3] Instituto Superior Técnico / INESC-ID Isabel.Trancoso@inesc-id.pt

Abstract. Current Computer Assisted Language Learning (CALL) tools, which have largely remained unchanged over the past few years, are often inadequate for attaining fluency in a second language. The goal of this work was to build a CALL tool based on bilingual eBooks, following the EPUB3 format, which aims to develop a wide array of necessary competences required from language learning. The tool should have an intuitive user interface, include the possibility of retrieving word definitions, and also include a set of gamified exercises.
The project comprised the development of a Python script which took the raw inputs (parallel texts in L1 and L2 and pre-recorded audio in L2) and merged them into a single structured eBook document, complete with dynamic features. These features offer the user an improved control over the audio playback (for instance, its speed), and are not included by default in the target platform, Apples iPad in this work. The script also retrieves, from an open-source API, dictionary definitions for every word in the eBook and includes them in the final document. Additionally, the project also comprised the development of a Javascript application that implements the gamified exercises included in the eBook, without relying on online external resources. This solution was required, since the chosen eBook format specification does not allow for online content to be fetched from inside the ebook.
A preliminary user study was conducted with a demonstration eBook generated by this system in order to improve its features and understand the expectations users would have for the eBook. Although positive feedback reassures the usefulness of this application, the demand for features that require deeper analysis of the dual language book raises concerns over the viability of the chosen reading platform, Apples ibooks application, and the EPUB3 format versus implementing the solution as an independent, standalone mobile application.

1 Introduction

Effects of globalization in recent years have created a demand for bilingual and multilingual individuals. Several languages such as English and Spanish have become highly sought after by employers for their value in international communication, the tourism industry and in trade. Other emergent languages, such

as Mandarin Chinese, are also sought after for the value it is perceived of them from times to come. Migration has also influenced these demands as it shapes countries into multilingual and multicultural societies, and so has the Internet with its accessibility to readily connect individuals anywhere around the globe. It is for these reasons that people and countries as a whole seek to prepare themselves to deal with the challenges of a modern bilingual world by instructing themselves and their youth in a second language (L2).

Despite the vast accessibility to foreign media, which supplies a plethora of means of study and opportunities for enriching oneself with new knowledge of the language to advanced students, most of the tools available to beginner and intermediate L2 learners have remained unchanged over the past few years. These instruments are generally textbooks, audio compact discs and other similar media, which allow students to study and learn some new insights into the L2 on their own. Proficiency in L2, however, comes from achieving a degree of fluency, complexity and accuracy. These are attained from acquiring a competence in understanding formulaic expressions and knowledge of the L2 grammatical rules [11]. For a student to obtain these skills, studies suggest that instruction should revolve around an extensive input in the L2, as well as focus on the meaning of its formulaic expressions, but also give the student the possibility of output [5].

This paper describes the development and evaluation of a tool to generate multimodal, bilingual ebooks in EPUB3 format, which in addition include embedded gamified exercises to train L1 speakers onto L2 language.

2 Background

2.1 Theories for language learning

Computer-assisted language learning CALL is an approach to language learning and teaching which includes computer technologies as a means to promote educational learning through a substantial interactive element. It can be used to help students attain fluency, complexity and accuracy in oral skills, literacy and formulaic expressions [6, 8].

Owing to audio features, CALL programs provide their users with a profusion of different ways in which they can practice their listening skills. For instance, videos supporting L1 or L2 subtitles or explanatory notes, along with an option to quickly and easily control its playback can improve both immediate comprehension and acquisition [3].

Also, thanks to ASR (automatic speech recognition) systems, L2 learners are no longer restricted to practice speaking with a partner, but are able to talk to a computer to evaluate their performance, or practice short dialogues [12, 9, 1].

For writing skills, L2 learners can use CALL programs to enrich their vocabulary and benefit from the now widespread spell and grammar checking systems. Such systems enable L2 learners by quickly giving them feedback which develops the L2 writing accuracy of the learner.

Gamification Gamification is the concept of applying game design mechanics, techniques and game principles to improve user engagement, productivity and motivation. Games require the players to learn, even if just the rules of the game. Upon mastering the rules, the player has the creative freedom of applying and navigating the rules. When developing a new game, the game designer must take into consideration how to convince the player to learn how to play the game and use good methods and principles to ensure that it is a fun experience [10].

2.2 Text and audio processing algorithms

Parallel text alignment When dealing with large parallel texts, such as bilingual eBooks, the task of alignment can become challenging. When a text is translated, particularly when considering how different two languages can be in terms of structure, grammar and syntax, some sentences may be split, merged, reordered, deleted or inserted by the translator.

Alignment may be executed at different levels. Some texts may benefit from paragraph alignment, while others from sentence alignment or even word alignment. Regardless, alignment methods are often categorized as either statistic or lexicon-based.

One such statistic approach is that of Gale and Church [7]. The motivation for their model came from the observation that longer regions of text tend to have long translations, and shorter regions have shorter translations, finding that the correlation between the length of a paragraph, measured in number of characters, between the original text and its translation was exceedingly high.

Lexicon-based alignment approaches use associative measures. Most basic lexical level methods use the Dice-coefficient, which is a statistic used for comparing the similarity of two elements, to keep co-occurrence scores for each word pair, and the total independent occurrences for those words in each of their respective texts. Then it calculates the probabilities of alignment and selects the sentence combinations of maximum likelihood.

A combination of both statistic and lexicon-based alignment methods is the approach of most modern parallel text aligners, such as the one used in the present work, Hunalign, an open-source parallel text aligner that aligns text on the sentence level.

Forced alignment and speech recognition systems Forced alignment systems attempt to match the words of a given input text to those of an input audio file. The input text is mapped into a phone sequence by using a pronunciation dictionary, and phone boundaries are determined by algorithms such as the Viterbi algorithm.

Current forced alignment systems will usually be composed of two parts, a training stage and an alignment stage. During training, the system learns the correspondence between phonemes (usually including left-right context) and audio-derived features.

When attempting alignment, the system aligns the new speech with its acoustic models through the Viterbi algorithm to produce time stamps for each word

in the given audio file. The forced alignment process is identical to that of ASR, except that a phonetic transcript is known [4]. Thus, in ASR the Viterbi algorithm is used to find the most likely path through the phoneme HMM and from that the system then generates the resulting word string.

In this work we used a slightly different approach proposed in [2] where both text and audio are converted into phoneme sequences and then a dynamic programming algorithm based on the edit distance aligns both sequences to obtain the alignments. This method has the advantage over standard forced alignment techniques in that it can align very long audios to texts without the usual memory burdens of decoding long audio files.

2.3 EPUB 3.0.1

EPUB is a widely adopted format for digital books, eBooks, due to its potential to comprise complex layouts and interactivity in the eBook. This is achieved by the use of the latest HTML5 standard by EPUB 3.0.1, which allows EPUB publications to contain video, audio and interactivity through scripting, much like modern web pages. EPUB reading systems paginate the content dynamically, adapting it to the display, rather than having the reader zoom in or out a fixed format page, as is common with other applications such as PDF. This behavior is not forced in any way, however, allowing for specific content that would benefit from a zooming and panning functionality to be manipulated in such a way.

Being comparable to a web page EPUB is, consequently, a simple ZIP archive file that contains the HTML files, CSS style sheets, metadata and any other assets necessary for EPUB reading applications or devices. One such file contained in the EPUB is the Package Document, which is a unique file in the publication. It specifies all documents, images and other content that make up the publication, including the ordering of the pages of the eBook.

Most EPUB readers support the use of the synchronized multimedia integration language (SMIL). SMIL allows the synchronization of the transcript of a publication with a corresponding recorded audio file. It is embodied by a timings file that matches marked segments of text to portions of the audio file linked to each page.

3 Specification and Design

3.1 Specification

The objective of the software system developed in this project is twofold. First, it must create the eBook and its user interface, complete with a dictionary mode. The user interface must allow control over both playback and speech rate. Secondly, it must include gamified exercises in the eBook. It is intended that the creation of the EPUB formatted eBooks be fully autonomous, requiring no external resources other than the text given in two distinct languages, its corresponding audio file and parallel text alignment and forced alignment tables (which in turn could be obtained automatically using other automated tools).

Due to the use of the EPUB format, any resource or data required by the eBook must be self-contained, as it is not possible for the eBook to fetch any online resource. Static resources available online required by the eBook must be fetched by the eBook generator script.

Regarding the specifications of the parallel book section, the eBook must provide the user with a sentence highlighter which establishes a correlation between the sentence being read by the narrator and the corresponding text segment in the second language. The user interface must allow the reader to seamlessly control audio playback, through commonplace play and pause functions and pace and flow control processes as well.

The dictionary mode must be implemented separately as the specifications for this project render the reader's own dictionaries inaccessible. This, however, enables future development to disambiguate the word definitions. The dictionary mode is expected to be able to return the definition of every word in the eBook that has an entry in the given source. A few select words are to be underlined at all times, meaning their definitions may be accessed without the use of dictionary mode. These correspond to the words deemed most important and relevant for the readers learning process.

Regarding the gamified exercises, there are a few restrictions imposed on their design. The games revolve around the text in the eBook, requiring the user to translate words into the second language given their context in the original transcript. The games are also expected to utilize gamification elements to bolster the users motivation and engagement levels.

3.2 Design

The eBook is to allow the user to play the audio starting from any given sentence in the eBook. To do this, the user should tap the text corresponding to the sentence he wishes to start from. In addition, the eBook is to provide means that help better comprehend the audio. It is the responsibility of the EPUB generator script to include this functionality in all eBooks. On running, this first component expects a number of eBook independent resources, namely icons, images, text Fonts, scripts and optionally a *list of important words* to underline in the eBook that is going to be processed. The images and icons are needed to make a small graphical user interface, which is required to be non intrusive, so as to not disrupt reading.

In addition to the dictionary mode, the user interface includes tools which ease reading along with the audio. These functions are displayed in figure 1. In order, they are play/pause, rewind, repeat, one-by-one mode (in which only one sentence is played, at the end of which the audio is paused), slow/normal pace audio and the dictionary mode. The *important word list* is optional. If none is present, important words are selected from the text on an occurrence frequency basis. These words are to be underlined by default on the eBook, and pressing them will bring up a short definition. Regardless of importance, however, every word in the eBook will have a corresponding definition which can be accessed through the dictionary mode in the user interface.

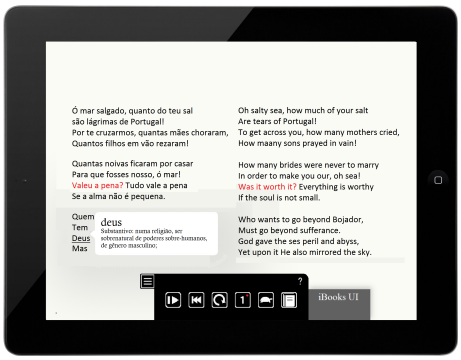


Fig. 1. Graphical user interface and word definition.

The gamified exercises module is placed in the last page of the eBook. Upon loading this page the user is greeted with two games to choose from, *Fill in the blanks* and *Word blitz*. The execution of these games is similar, but their presentation is fundamentally different. In both games, the eBook's text is used to create the games. Sentences or whole pages are selected, and from them, a random word or set of words is replaced with a blank which the user is then prompted to fill with the correct missing word, with the assistance of the corresponding audio.

In the first game, *Fill in the blanks*, whole book pages are used, and several words are removed from it. Initially hidden, one by one, the sentences are then revealed to the user, and the corresponding audio track is played. Should a revealed sentence have one or more of these blanks, execution is halted until the user fills all blanks. When the page is completely revealed and all game actions are exhausted, a score is given to the user depending on how they fare. This design results in prolonged games which tests a subject's consistency.

The second game, *Word blitz*, however, uses a single sentence with a single blank per game, resulting in quicker, more relaxed games with which the player may spend any amount of time they see fit. This will both directly and indirectly aid them in *Fill in the blanks*, as playing *Word blitz* translates not only into acquiring a greater closeness to the eBook, but also into hints that may be used to help achieve higher scores.

The final feature, the hint system, is a gamification tool that rewards users for their time spent in self-assessment. By design, the hint system is only usable during *Fill in the blanks*, but hints may be earned by participating in either game. As players correctly identify the missing words in the games, a progression bar fills. Once entirely full, they earn a hint. In order to use the hint, for the current sentence, a blank has to be selected through the hint menu's drop down list. Once the blank is chosen, two types of hints are available, *Check correctness*, which, without detriment to their scores, reveals whether the text written in the blank is the correct solution; and *Partial solution*, which writes an initial portion

of the word in the blank, useful for when the user has trouble ascertaining the exact word in the audio playback.

4 System Implementation

4.1 EPUB Generator

Upon executing the EPUB generator, it loads all resources required for creating the eBook that necessitate no editing, such as the audio files, scripts, images and icons. Then, it loads the parallel alignment and forced alignment tables and uses them to make a word count. Then, depending on the user, a list of words considered important in the eBook is either pulled from a list, or generated with the word count. Should the latter be selected, two parameters decide which words are added to the important word list: a count threshold, and a word length (in characters) threshold.

Subsequently the EPUB generator script loads the text for both languages and, page by page, merges their content together into a single file, complete with header declarations and user interface elements. This process continues by using the parallel alignment table to create identifiers for sentences or sentence groups. These identifiers will later allow the SMIL emulator to highlight corresponding sentences in both languages as they are read by the audio playback. Finally, the input forced alignment table is read and converted from its word by word representation into groupings delimited by the identifiers. A script element is then added to the page containing a data structure which lists all SMIL timing information for that specific page. Once all pages have been processed, the dictionary manager is loaded, and its definitions incorporated into the eBook. This module consists of a class definition for a dictionary manager and database. The methods defined within the class allow for querying Wiktionary's MediaWiki API for new definitions, and parsing the received response. The parser was implemented specifically for this application, as no other parsers seem to exist for Wiktionary.

Finally, the EPUB generator script concludes the task by creating a final HTML page for the gamified exercises, as well as the Package Document.

4.2 SMIL emulator

This module borrowed its backbone from the now deprecated rb_smil_emulator Javascript module created by Alberto Pettarin. rb_smil_emulator allowed for mirroring the operation of iBooks's built-in SMIL interpreter. Taking this approach was necessary, as each of the eBook's pages must be bound to a single SMIL file, which in turn must be associated to a single audio file. Given there is no support to control the pace of the audio when using SMIL, it was deemed necessary to use two separate tracks, one for the regular audio track, and a slowed down version of the same track. This demands the creation of a separate SMIL table, which in turn rules out the possibility of using iBook's built-in reader.

The final implementation consists of a Javascript code able to not only simulate the iBooks SMIL interpreter but also provide extra features otherwise not

available. Such features include ways to repeat or rewind the audio playback, set a multiplier to the SMIL data and switch the audio track to reproduce the toggle effect for slowing/speeding up the audio playback, and enabling dictionary mode during which no audio playback is allowed.

4.3 Gamified exercises

The first game, *Fill in the blanks*, uses a version of the SMIL emulator mentioned beforehand. It is heavily modified, includes only the normal pace audio file, and does not register touch events. Instead, the task of moving on to the next sentence of repeating the audio of the active sentence falls upon function buttons. The source of contents for this game is a page in the book, randomly selected by the creation tool. The selection of words to be removed (i.e. the blanks) is made before the game begins, and these words are selected randomly within the boundaries of three parameters. Firstly, the distance between two words must be within limits specified by the designer. The final parameter is the minimum word length. Only words greater in character count than this parameter are used in calculating the distances of the previous two parameters. This process uses a cumulative distribution function to ensure that the randomly selected words meet this criteria. All selected words are then stored in a list for use with the hint system and scoring.

On the second game, *Word blitz*, a page is also selected at random. Upon acquiring the number of segments in the selected page, a text segment is chosen randomly, and a single word in it is selected to be the removed. Both these selections have criteria, as in *Fill in the blanks*. Eligible words are those with a character length greater than a given threshold, and a minimum threshold for the segment length, in words, is also used.

5 Results and Evaluation

Feedback from a test group supported the development of the system. The test group comprised of seven individuals from both the United Kingdom and the United States of America who had either negligible or no prior experience with the Portuguese language. This group indicated which features are of their preference and offered further development suggestions. This option was chosen in lieu of a comparison of a test group using the eBooks developed in this project and a control group using a different form of learning aids. The reason for this being that when using small test groups, it is inconceivable that both groups integrate individuals of comparable learning propensity and identical L2 backgrounds. Consequently, it would be impractical to analyze and compare results based on the two test groups.

It should be noted that all participants were shown the same short demonstration eBook. The L1 for the eBook being English and the L2 target language Portuguese, meaning all audio and games are featured in Portuguese. While the text pages are the same for all participants, due to the random nature of the

gamified exercises, no two participants were given the exact same games to play. Even those who coincidentally chose the same game pages did not experience the same challenges, owing to the fact that the blanks are generated dynamically.

Furthermore, a group of fourteen Portuguese individuals participated informally in the user study. Their feedback was taken into consideration for matters such as usability, user interface and user experience development. Their experiences could not be included in the results since they were given the same demonstration eBook, where the target language matched their mother tongue.

5.1 Survey hypotheses

1. **Audio support promotes second language comprehension in dual language books.** This is the main idea behind this work. Every other function expands on this concept and as such, it is considered the focal point of the survey.
   (a) **The highlighting aids reading along to the audio playback.** It is expected that the offering of a simple visual queue to the reader facilitates reading along to the audio playback.
   (b) **The highlighting allows for an easier correspondence of phrases in either language.** The anticipated effect is that of aiding in the analysis of the meaning of the corresponding highlighted sentences. Knowing the corresponding blocks of text is expected to enable a finer interpretation of each individual word.
   (c) **Control over audio speed and playback facilitates the acquisition of phonetic characteristics of the L2.** This encloses all other implemented functions, such as the UI, slower playback speeds, and playback control. One example is that of the replay button, as it allows maintaining the text in plain view while the audio is being repeated, as often as one would like, without having to press the transcript, thus blocking their own view.
2. **Gamified exercises motivate and gratify second language learning.** Gamification, when well designed, should appeal to its target audience and influence their behavior concerning their tasks, raising levels of engagement, motivation and sense of gratification.
   (a) **The hints system drives users to dedicate more time to their studies.** This acts as the game's currency. It correlates directly with a tangible benefit the students may use at will. It rewards those who carefully study the book beforehand, and eases any possible feeling of stress derived from a situation where one correct answer is all that is holding back the student from a perfect score.
   (b) **The scoring system encourages players to commit themselves for better results.** As briefly touched upon on the previous hypotheses 2(a), the scoring system should drive players who appreciate gaming self-elements to raised levels of engagement in an attempt to pass the books games with a perfect score.

5.2 Informal discussion

Regarding current functionalities, participants criticized several features. Amongst those were the lack of a toggle or feedback and responsiveness when using certain functions, word definitions being poor and the original audio pace being too quick. In addition, some concern regarding the effectiveness of the eBook for language learning arose due to the fact that these books are very poor in regards to learning correct grammar.

Ideas offered for future development included a word for word highlighter and single word audio playback, the inclusion of IPA phonetic notation in the dictionary and further functionalities to control audio pace.

5.3 Survey results

Participants were surveyed on their perceptions of both the dual language book as well as the gamification elements included in the exercises. All propositions are answerable on a scale of one to five, where one represents a complete disagreement and five a strong agreement.

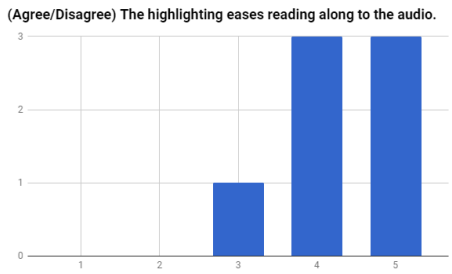


Fig. 2. eBook question 1. Mean: 4.29.

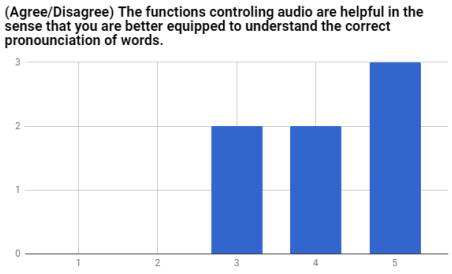


Fig. 3. eBook question 3. Mean: 4.14.

Results regarding this and other questions were largely favorable, indicating a general consensus that the implemented functions are well suited to meet the formulated hypotheses.

Although results for the gamification elements segment of the survey were mostly positive, the hints system pales in comparison to the other features, indicating that this feature requires additional development. Moreover, when filtering the results with respect to demographics, focusing on individuals who did not speak Portuguese, questions 1 and 3 are altered significantly, with predominantly neutral responses.

6 Conclusions and Future Work

The hypotheses formulated in the preceding chapter were met to a satisfactory degree by the results of the user study. The hypothesis 2(a), regarding hints

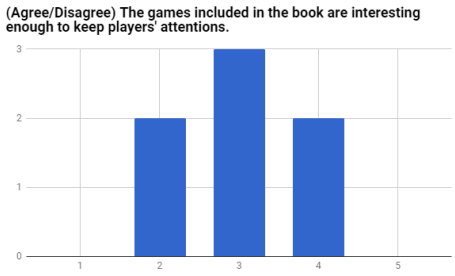


Fig. 4. Gamification elements question 1. Mean: 3.00.

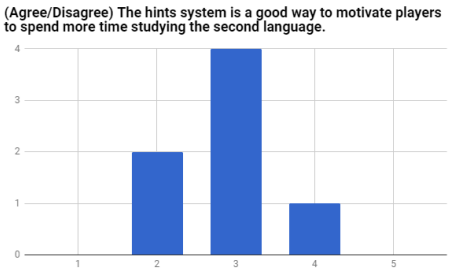


Fig. 5. Gamification elements question 3. Mean: 2.86.

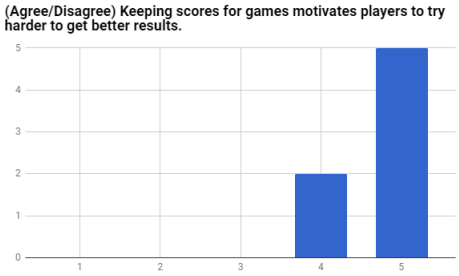


Fig. 6. Gamification elements question 4. Mean: 4.71.

system, however, fell short in delivering the intended gamification element of currency and should be improved upon.

Considering the implementation phase and features participants requested for during the user study, Apple's eBook reader, ibooks, fell short of expectations. It was not possible to use SMIL dynamically as Javascript code that altered the HTML pages in any form ran sluggishly and, despite tailoring a SMIL emulator to suit the eBooks's requirements, timing issues plagued every function call that depended on audio playback.

Considering that the above issues were not at all present when testing on web browsers such as Google Chrome and Internet Explorer, one assumes that ibooks is not prepared for dynamic content not created through its own built-in functionalities. These processes are, however, very basic in nature and unsuitable for projects that require even the least amount of complexity.

Future work should explore options beyond ibooks. As of a year prior to the writing of this article, no other reader applications existed which offered a greater array of developmental tools than ibooks. Perhaps an eBook reader could be designed to catalog and display the eBooks. Having control over the reader application would allow for designs of greater complexity, including features unavailable in other readers, such as loading auxiliary files to meet data requirements for simpler software architectures. Also unavailable in ibooks, internet access could allow for recommendations of eBooks given each students'

GotIt!: uma plataforma de *crowdsourcing* para entreajuda quotidiana de cegos e idosos

Hugo Paredes[12][0000-0002-4274-4783], Hugo Fernandes[12], António Marques[1], Arsénio Reis[12] e João Barroso[12][0000-0003-4847-5104]

[1] INESC TEC, Portugal
{hparedes,hugof,ars,jbarroso}@utad.pt
[2] Universidade de Trás-os-Montes e Alto Douro, Portugal
amarques@utad.pt

Abstract. Este artigo tem como objetivo apresentar e analisar os principais desafios no desenvolvimento de uma plataforma de *crowdsourcing* para inclusão e apoio social. O sistema GotIt! foi projetado para ser uma plataforma genérica para a criação de cenários de *crowdsourcing* inclusivo. Para a demonstração do sistema foi escolhido um cenário em que cegos e idosos se entreajudam: os primeiros para ultrapassar barreiras diárias e conseguir aceder a informação apenas disponível sob o formato visual; os segundos para combater o isolamento e manterem-se socialmente ativos, apoiando quem necessita de ajuda.

Keywords: crowdsourcing, cegos, idosos, inclusão, entreajuda.

1 Introdução

A ascensão das tecnologias pervasivas relançou os problemas de acesso universal numa perspetiva mais ampla, abrindo novas oportunidades para a inclusão digital. A capacidade de interligar múltiplos utilizadores com modos de acesso diferentes e permitir a sua interação e colaboração, generaliza a necessidade de migrar do acesso universal para a colaboração universal. O poder das ferramentas colaborativas, associado à massificação da utilização da Internet e à ubiquidade da sua utilização tem alavancando a colaboração em massa e a divisão do trabalho em micro tarefas que podem ser executadas por qualquer um. Esta inteligência coletiva pode ser explorada de várias maneiras como o *crowdsourcing*, computação humana e a computação social.

Neste artigo é explorada a utilização de *crowdsourcing* móvel no domínio da inclusão e apoio social, promovendo a entreajuda e o envolvimento em comunidade. Para tal foi tido como caso de estudo o quotidiano dos cegos e a necessidade, mesmo que esporádica, de acesso a informação exclusivamente disponível sob a forma visual. Neste cenário têm sido propostas diversas soluções que recorrem à visão por computador para processamento automático de imagens ou vídeo. No entanto, em determinadas circunstâncias, estas soluções revelam-se pouco fiáveis face às condições do ambiente. Por seu lado, a computação humana revela-se como uma solução fiável, podendo funcionar como potenciador de inclusão social, desde que provido de tecnologia adequada

às necessidades dos utilizadores e de mecanismos capazes de promover a sua participação. Esta, pode ser uma oportunidade de inclusão (digital e) social dos idosos, que, atualmente vivem em isolamento, necessitando de um papel mais ativo na sociedade, podendo desta forma auxiliar os cegos no seu quotidiano.

A plataforma GotIt! foi desenvolvida para fornecer o suporte tecnológico necessário à entreajuda entre cegos e idosos. Está dotada de duas aplicações móveis, desenhadas especificamente para os dois públicos alvo e de um componente servidor que efetua a distribuição das solicitações e agregação das contribuições. As aplicações móveis têm um desenho minimalista resumindo as interações com o utilizador ao estritamente necessário, através de fluxos simples e intuitivos. Este trabalho tem como objetivo apresentar a plataforma GotIt! e analisar os principais desafios no desenvolvimento de um cenário de utilização da plataforma de *crowdsourcing* para inclusão e apoio social.

Este artigo está organizado em 5 secções: na secção seguinte é apresentado o enquadramento para este trabalho, incluindo o atual estado da arte no domínio das ferramentas colaborativas e de *crowdsourcing* para inclusão e apoio social. O sistema GotIt! é introduzido na secção 3, sendo apresentada uma perspetiva genérica do sistema e dos seus principais componentes e atores. Na secção 4 é exposta a implementação de um caso de aplicação do sistema GotIt! aplicado à entreajuda entre cegos e idosos. O artigo termina com a seção 5 onde são apresentadas as conclusões e perspetivas de trabalho futuro.

2 Enquadramento

A ascensão das tecnologias pervasivas relançou os problemas de acesso universal numa perspetiva mais ampla, abrindo novas oportunidades para a inclusão digital. A capacidade de interligar múltiplos utilizadores com modos de acesso diferentes e permitir a sua interação e colaboração, generaliza a necessidade de migrar do acesso universal para a colaboração universal.

A riqueza da interação permite normalmente reunir nos sistemas colaborativos utilizadores com diversas necessidades, habilidades e interesses. Como resultado, estes sistemas, designados de *groupware*, têm como requisito favorecer as diferenças, oferecendo suporte ao acesso por pessoas com diferentes níveis de alfabetização, diversos níveis de capacidades com ferramentas tecnológicas e com aptidões e/ou necessidades especiais que potenciam a execução de certas atividades [1]. O desenho de aplicações colaborativas necessita de identificar os aspetos práticos, confiança e possibilidades de interação contínua, integrando a colaboração síncrona e assíncrona. Um dos passos fundamentais para permitir a acessibilidade universal é a definição inteligente do "perfil" do utilizador e a semântica de serviços disponibilizados [2]. Formas alternativas de apresentação, suportando colaboração multimodal constituem também um requisito para a adaptação destes sistemas aos emergentes ambientes ubíquos de colaboração, fornecendo meios para colaborar, resolver problemas e participar de uma forma que é benéfica e significativa para o grupo e para cada individuo [3]. As interfaces têm o

potencial de expandir a acessibilidade a diversos utilizadores não especialistas, de diferentes idades, níveis de habilitações, estilos cognitivos, deficiências sensoriais e motoras, língua materna, ou mesmo doenças temporárias [4].

O poder das ferramentas colaborativas, associado à massificação da utilização da Internet e à ubiquidade de utilização tem alavancando a colaboração em massa e a divisão do trabalho em micro tarefas que podem ser executadas por qualquer um. As estatísticas revelam que em 2015 estava conectada através da internet cerca de 45% da população mundial, num total aproximado de 3.2 mil milhões de pessoas[1], representando uma enorme força de trabalho se tecnologias pervasivas puderem ser integradas nas suas tarefas quotidianas. A inteligência coletiva pode ser explorada de várias maneiras como o *crowdsourcing*, computação humana e a computação social. A ideia de usar pequenas unidades de trabalho humano num algoritmo foi introduzida pelo TurKit [5] demonstrando o conceito em fluxos de trabalho simples com melhorias no processo iterativo. Posteriormente, o conceito foi expandido a fluxos complexos, como o CrowdForge [6], Turkomatic, Mobi [7], Legion [8] e Scribe [9].

A computação humana pode fornecer informações em tempo real com extrema precisão para problemas reais [10]. Questões de desenho, barreiras cognitivas, problemas de acessibilidade e precisão na autoria de meta-dados são algumas variáveis que devem ser analisadas, incluindo as pessoas com deficiência, desde a fase inicial do processo de desenvolvimento da computação social [11]. As melhorias na acessibilidade da colaboração têm sido vistas como uma abordagem fundamental para melhorar a experiência de navegação Web para as pessoas com necessidades especiais [12]. O ecossistema de computação móvel apresenta diversas oportunidades de *crowdsourcing*. O uso ubíquo de telemóveis e tablets num conjunto diversificado de ambientes de trabalho revela requisitos que variam continuamente. Neste tipo de configuração em que multidões humanas podem agir como operadores cognitivos executando tarefas de computação em qualquer lugar e a qualquer hora, é essencial a partilha de conhecimento de forma rápida e eficiente. Os recentes serviços comunitários, como Be My Eyes[2], permitem que voluntários de todo o mundo possam ajudar as pessoas cegas através de uma videoconferência a partir de um telemóvel. O Viz Wiz [13] é uma aplicação móvel que permite que os utilizadores cegos façam perguntas para a "multidão" ou para os seus amigos. Outro exemplo é o Friendsourcing que permite que amigos remotos possam responder a perguntas visuais para os cerca de 39 milhões de utilizadores cegos em todo o mundo [14]. A digitalização colaborativa de livros e a sua transcrição por voluntários mostram uma possibilidade de investigação para melhorar a qualidade de vida dos cegos (por exemplo, Bookshare [15]).

Os sistemas movidos por multidões podem combinar a computação com a inteligência humana, a partir de grandes grupos de pessoas interligadas e coordenadas na internet. Estes sistemas híbridos permitem o desenvolvimento de soluções que nem as multidões nem a computação poderiam suportar sozinhos [16]. Uma das perspetivas para a

[1] De acordo com os relatórios da International Telecommunication Union (ITU) http://www.itu.int/en/ITU-D/Statistics/Pages/default.aspx

[2] http://bemyeyes.com/

inclusão da perceção humana usa a computação humana no processo de aquisição, fusão e prospeção de dados. De igual modo, o *crowdsourcing* móvel tem ganho destaque como uma solução viável para resolver problemas em grande escala [17]. Apesar das soluções existentes na área do *crowdsourcing* móvel, ainda há um longo caminho a percorrer no que respeita as questões de socialização, envolvimento em comunidade e aplicações de grande volume de dados.

3 O Sistema GotIt!

A colaboração acessível tem-se afirmado como uma abordagem fundamental para a resolução de alguns problemas de acessibilidade. Design, barreiras cognitivas, problemas de acessibilidade e criação precisa de meta-dados são algumas variáveis que requerem uma análise humana associada ao processo computacional, inviabilizando o processamento isolado pela máquina (dada a complexidade dos relacionamentos a processar) ou pelo homem (dada a quantidade de dados a processar). A conjugação de ambas assume-se como uma necessidade para algumas destas atividades é já uma realidade. A abrangência desta colaboração pode ser alargada às massas (*crowds*) pelo uso ubíquo da tecnologia permitindo a participação em qualquer hora e em qualquer lugar. Neste tipo de configurações, as massas podem agir como operadores cognitivos executando tarefas de computação e a partilha de conhecimento de forma rápida e eficiente.

O sistema GotIt! tem a sua gênese conceptual neste contexto, sendo baseado num modelo impulsionador da participação, interação e apoio mútuo. O sistema promove a integração e ajuda mútua entre as pessoas, independentemente das suas necessidades especiais ou limitações, conjugando computação humana com os recentes avanços das tecnologias da informação e comunicação, resultando de um modelo orquestrado de mecanismos de inteligência coletiva com iniciativas de inclusão social.

Genericamente pode ser visto como uma arquitetura cliente-servidor (**Fig. 1**). Os clientes são preconizados pelas aplicações móveis disponíveis e adaptadas a cada um dos atores do sistema. Por seu lado a componente servidor atua como mediador da comunicação entre consumidores e fornecedores de serviços, tendo como responsabilidades o registo, distribuição e coordenação dos pedidos.

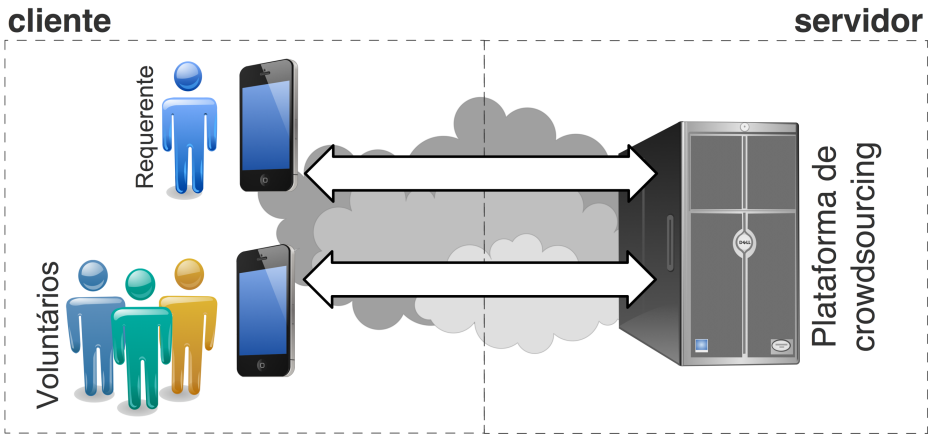


Fig. 1. Arquitetura do sistema GotIt!

O funcionamento deste sistema pode ser sistematizado em 8 fases:

(1) Necessidade de apoio;
(2) Recolha de dados necessários ao pedido de ajuda;
(3) Pedido de ajuda;
(4) Distribuição do pedido por voluntários registados;
(5) Aceitação/rejeição de pedido de ajuda;
(6) Ajuda;
(7) Agregação das respostas ao pedido de ajuda;
(8) Resposta ao pedido de ajuda.

No processo são também identificados três atores: requerente de apoio; voluntários e sistema. O requerente de apoio é quem inicia o processo quando tem necessidade de apoio. As fases 1-3 são realizadas por este ator. Na fase 4 intervém o sistema, distribuindo o pedido pelos vários voluntários ativos no sistema. Os voluntários são os intervenientes nas fases 5-6, cabendo novamente ao sistema a execução das fases 7-8.

4 Protótipo do Sistema

O sistema GotIt! foi projetado de forma abstrata para que possa ser instanciado em diversos cenários de aplicação. Apesar do seu desenho genérico, é expectável que a sua aplicação preferencial sejam situações de inclusão, disponibilizando um canal prioritário de apoio a pessoas com necessidades especiais e idosos. Neste sentido, o cenário de aplicação aqui ilustrado reflete este requisito, aplicando o sistema GotIt! à entreajuda entre cegos e idosos.

6

4.1 Cenário de aplicação

O envelhecimento das sociedades exige medidas para prevenir o isolamento social, podendo a participação em iniciativas de *crowdsourcing* motivar a socialização e evitar o isolamento de idosos. No seu quotidiano os cegos enfrentam diversos desafios, principalmente quando a informação disponível está restrita ao formato visual. Em muitas situações a tecnologia pode apoiar os cegos nas suas tarefas diárias, estando, no entanto, limitada à capacidade computacional. Consequentemente, o apoio prestado por outras pessoas pode ser crucial para ultrapassar determinados obstáculos do dia a dia. A disponibilização de ferramentas tecnológicas acessíveis permite a participação ativa dos idosos nesta ajuda quotidiana aos cegos, representando também ela uma iniciativa de participação social para os idosos.

O cenário apresentado representa uma possível instanciação para o sistema GotIt!. Neste caso particular, os requerentes de apoio são os cegos, sendo o papel de voluntários desempenhado pelos idosos. As 8 fases que representam o funcionamento do sistema são representadas no *storyboard* apresentado na **Fig. 2.**

Fig. 2. Storyboard

O processo inicia-se com a necessidade de um cego de reconhecer um objeto ou informação disponível apenas em formato visual. Nesse momento inicia a sua aplicação para fazer um pedido de ajuda. A aplicação disponibiliza automaticamente o modo fotografia em que o cego pode apontar para o objeto e fotografar. A imagem pode então ser enviada para o sistema GotIt!. Ao ser recebida, a imagem é distribuída por vários voluntários registados, enviando uma notificação para a sua aplicação. Para aceitarem o pedido de ajuda, os idosos apenas necessitam de abrir a notificação que automaticamente lhes apresenta a imagem e um botão para descrever a imagem usando reconhecimento de voz. A descrição é guardada em texto e enviada para o sistema, que agrega as respostas dos vários voluntários e constrói a resposta a enviar ao cego. A resposta é

enviada, sendo a aplicação do cego notificada da recepção da resposta do pedido de ajuda.

Nesta instanciação, o sistema GotIt! é suportado por duas aplicações móveis: uma para o cego e outra para os idosos. Estas aplicações foram desenvolvidas tendo em consideração os requisitos destes dois grupos de utilizadores. A sua implementação é descrita na subsecção seguinte.

4.2 Implementação

A implementação do cenário de aplicação do sistema GotIt! seguiu a arquitetura genérica apresentada na **Fig. 1**. Conforme descrito, foram desenvolvidas duas aplicações cliente: uma para os cegos pedirem apoio; outra para os idosos prestarem ajuda. Os protótipos das aplicações foram desenvolvidos para o sistema Android, tendo sido utilizado o Android Development Studio.

A metodologia de desenvolvimento seguiu uma abordagem centrada no utilizador, em que inicialmente foram desenvolvidos protótipos de baixa fidelidade que foram apresentados a um grupo de três utilizadores idosos e dois cegos. A apresentação dos protótipos de baixa fidelidade aos utilizadores cegos passou pela descrição dos protótipos de baixa fidelidade pelos investigadores. Com base no feedback recolhido, foram implementados os protótipos funcionais.

O desenho da interface das aplicações teve como principal premissa minimizar a interação com o utilizador. Deste modo, o fluxo da aplicação para o cego pedir ajuda pode ser sintetizado em 4 passos (**Fig. 3**):

(1) Abrir aplicação
(2) Tirar fotografia
(3) Enviar fotografia e aguardar descrição
(4) Receber notificação e ouvir descrição

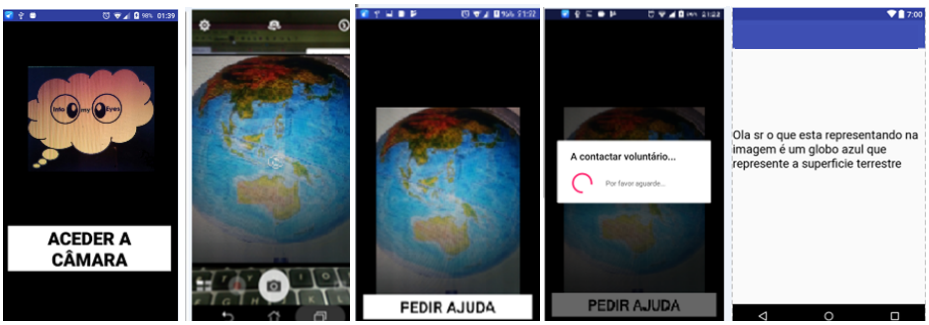


Fig. 3. Capturas de ecrã da sequencia de interações na aplicação do cego.

A aplicação para os idosos segue os mesmos princípios, sendo o seu funcionamento reduzido a 3 interações (**Fig. 4**):

(1) Receber notificação e abrir aplicação

(2) Descrever fotografia usando reconhecimento de voz
(3) Enviar descrição

Fig. 4. Capturas de ecrã da sequencia de interações na aplicação dos idosos.

A mediação da comunicação é assegurada por um componente servidor que recebe os pedidos de ajuda da aplicação dos cegos e distribui o pedido pelos vários voluntários registados. O serviço passa então para o modo de espera, em que aguarda a resposta dos voluntários. Por forma a assegurar uma resposta em tempo útil foi definido como tempo limite para a resposta 90 segundos. Passado esse tempo e não havendo nenhuma resposta a aplicação do cego é informada que não houve resposta para o pedido de ajuda. Caso contrário, são recebidas as várias respostas dos voluntários e preparada a resposta. Para o protótipo foi considerada a primeira resposta de um voluntário como a resposta valida. Em futuras implementações será considerada a análise das respostas e a agregação de resultados, tendo em consideração que será válida a resposta que consiga maior unanimidade por parte dos vários voluntários. Este modo irá requerer análise de linguagem natural, uma vez que as respostas são dadas pelos idosos utilizando reconhecimento de voz.

5 Conclusões e trabalho futuro

O artigo apresenta uma proposta para a colaboração inclusiva, concretizada no sistema GotIt!. A proposta visa criar uma plataforma tecnológica que permita a entreajuda e a inclusão social. Para a concretização do conceito é demonstrado um cenário de aplicação: a entreajuda entre cegos e idosos. O trabalho aqui apresentado reporta resultados preliminares da conceptualização e desenvolvimento de um protótipo que concretiza o sistema GotIt! com base neste cenário de aplicação.

A implementação de um demonstrador do conceito permitiu criar um ambiente que proporcionará a realização de testes com utilizadores, tendo em vista a validação do sistema e a análise dos seus impactos. O demonstrador foi implementado tendo em consideração as necessidades dos utilizadores, em particular o desenvolvimento de interfaces minimalistas que facilitem a interação de cegos e idosos.

Os próximos passos incluem a realização de testes de usabilidade e satisfação de utilizadores que serão realizados com grupos de utilizadores cegos e idosos usando o protótipo desenvolvido. A evolução e iterações seguintes do protótipo desenvolvido refletirão os resultados da avaliação efetuada e o feedback dos utilizadores.

Agradecimentos

Este trabalho é financiado pelo projeto "Sistema Integrado para Aumento da Autonomia de Cegos" através do Prémio Inclusão e Literacia Digital 2015, atribuído pela Rede TIC e Sociedade da Fundação para a Ciência e Tecnologia (FCT).

Referências

1. Alves, L. L., Valadao, D., Carneiro, C. A., Correia, M., and Braga, W. (2010). Accessibility and collaboration among semi-literate and illiterate groups. In Col- laborative Systems II-Simposio Brasileiro de Sistemas Colaborativos (SBSC-II), 2010 Brazilian Symposium of, pages 48–51. IEEE.
2. Lee, S., Ko, S. H., Fox, G., Kim, K.-S., and Oh, S. (2003). A web service approach to universal accessibility in collaboration services. In ICWS, pages 333–339.
3. Gross, T. (2003). Universal access to groupware with multimodal interfaces. In Proc. of the second Int. Conf. on Universal Access in HCI: Inclusive Design in the Information Society (June 22-27, Crete, Greece). Lawrence Erlbaum, Hillsdale, NJ, pages 1108–1112.
4. Cohen, P. and Oviatt, S. (2000). Multimodal interfaces that process what comes naturally. Communications of the ACM.
5. Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. (2010). Turkit: human computation algorithms on mechanical turk. In Proceedings of the 23nd annual ACM symposium on User interface software and technology, pages 57–66. ACM.
6. Kittur, A., Smus, B., Khamkar, S., and Kraut, R. E. (2011). Crowdforge: Crowd- sourcing complex work. In Proceedings of the 24th annual ACM symposium on User interface software and technology, pages 43–52. ACM.
7. Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., and Horvitz, E. (2012). Human computation tasks with global constraints. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 217–226. ACM.
8. Kulkarni, A. P., Can, M., and Hartmann, B. (2011). Turkomatic: automatic recur- sive task and workflow design for mechanical turk. In CHI'11 Extended Abstracts on Human Factors in Computing Systems, pages 2053–2058. ACM.
9. Lasecki, W., Miller, C., Sadilek, A., Abumoussa, A., Borrello, D., Kushalnagar, R., and Bigham, J. (2012). Real-time captioning by groups of non-experts. In Proceedings of the 25th annual ACM symposium on User interface software and technology, pages 23–34. ACM.
10. Takagi, H., Kawanaka, S., Kobayashi, M., Itoh, T., and Asakawa, C. (2008). Social accessibility: Achieving accessibility through collaborative metadata authoring. In Proceedings of the 10th International ACM SIGACCESS Conference on Com- puters and Accessibility, Assets '08, pages 193–200, New York, NY, USA. ACM.
11. Taylor, A. (2011). Social media as a tool for inclusion. Retrieved from Homelessness Resource Center website: http://homeless. samhsa. gov/Resource/View. aspx.
12. Wang, S., Pan, H., Zhang, C., and Tian, Y. (2014). Rgb-d image-based detection of stairs, pedestrian crosswalks and traffic signs. Journal of Visual Communication and Image Representation, 25(2):263–272.

Evaluation of Word Embedding Vector Averaging Functions for Biomedical Word Sense Disambiguation

Rui Antunes and Sérgio Matos

DETI/IEETA, University of Aveiro, 3810-193 Aveiro, Portugal
{ruiantunes,aleixomatos}@ua.pt

Abstract. The biomedical lexicon contains a large amount of term ambiguity, which hinders correct identification of concepts and reduces the accuracy of semantic indexing and information retrieval tools.
Previous work on biomedical word sense disambiguation has shown that supervised machine learning leads to better results than knowledge-based approaches. However, machine learning approaches require the availability of sufficient training data, and generalization performance behind the test data is not known. Knowledge-based methods on the other hand make use of existing knowledge-bases and are therefore mostly limited to the quality of such sources of information about concepts.
In this work, we used word embedding vectors to complement the knowledge-base information. We represent the context of an ambiguous term by the average of the embedding vectors of words around the term, and evaluate the impact of using word distance for weighting this average. We show how this weighting improves the disambiguation accuracy of the knowledge-based approach in a subset of the reference MSH WSD data set from 86% to 88%.

Keywords: biomedical word sense disambiguation, knowledge-based approaches, word embeddings

1 Introduction

Nowadays there is a huge amount of textual data. In order to keep up with all that knowledge, automatic text mining systems for extracting and retrieving information are mandatory. Several Natural Language Processing (NLP) steps have to be accomplished for properly extracting information from text. The most important step is Named Entity Recognition (NER) [1], which deals with the identification of concepts and associates them to knowledge sources, since the whole information extraction task is strongly dependent in the accuracy of the identified concepts. Due to the ambiguity of the human natural language, textual documents are replete of ambiguities, and so the Word Sense Disambiguation (WSD) is a crucial part of the NER task [2]. Particularly in biomedical texts much more terms have many meanings, making it harder to extract accurate information.

Biomedical concept disambiguation aims to remove ambiguity from biomedical documents. Its goal is to normalize the polysemic terms, attributing only one meaning to each ambiguous concept. Given the possible meanings of each ambiguous concept, the surrounding context is used to help inferring the correct meaning. Supervised machine learning techniques and Knowledge-Based (KB) approaches can be followed to solve the WSD problem [3]. Supervised learning algorithms currently achieve the best results, however they require annotated training data. On the other hand, KB approaches have also drawn wide interest [4], since these approaches are less dependent on training data, being strongly dependent on the quality of the knowledge sources. Moreover, the use of multiple knowledge databases has been proven to bring benefits to the problem of WSD [5].

Mikolov et al. [6] proposed a continuous-bag-of-words model architecture for deriving vector representations of words from large unlabeled corpora. These vector representations of words are known as neural word embeddings, or simply word embeddings. This is a recent technique that has been extensively used in several NLP tasks, namely for the WSD task [7, 8].

In the reference biomedical ambiguity MSH WSD data set [9], Jimeno Yepes [10] proposed a supervised biomedical WSD method using word embeddings, achieving a top accuracy around 96% with a Support Vector Machine (SVM) classifier. On the other hand, Sabbir et al. [11] proposed a KB approach that also uses word embeddings, achieving a best accuracy of 92% on the same data set.

In a previous work [12], we proposed a KB method for WSD achieving a top accuracy around 85% in a subset of the MSH WSD data set. In this work, we improved our previous KB method using word distances to weight word embeddings, achieving a best accuracy around 88% in the same subset. We calculate an embedding vector for representing each surrounding context of the ambiguous terms, making a weighted average of the word embeddings using different averaging functions. For the best of our knowledge, Iacobacci et al. [13] were the first to weight word embeddings according to its word distance relative to the ambiguous term. The word embeddings were calculated from around 15 million MEDLINE abstracts. Furthermore, we calculate embeddings vectors for concept textual definitions extracted from the Unified Medical Language System (UMLS) Metathesaurus [14], which were used to calculate cosine similarities between them and the embedding vectors of the surrounding contexts of the ambiguous terms. Association values between two concepts, calculated using the co-occurrences of MeSH terms in MEDLINE citations, were used to weight the cosine similarities. With this approach we were able to infer the most similar sense given the surrounding context of a specific ambiguous term.

2 Biomedical Ambiguity Data Set

Jimeno Yepes et al. [9] proposed a method to automatically develop a biomedical ambiguity data set using the UMLS Metathesaurus and the MeSH indexing

of MEDLINE abstracts. Using this method, they created the MSH WSD data set, which is currently the most used data set for evaluating biomedical WSD systems. The MSH WSD data set is composed by 203 biomedical ambiguous entities. Each possible meaning of the ambiguous terms has at maximum 100 instances, where each instance corresponds to a MEDLINE abstract containing the ambiguous term. From the 203 ambiguous terms, 189 only have two possible senses, 12 have three possible senses, and the remaining 2 terms have four and five possible meanings.

In this work we only considered a subset (191 terms) of the MSH WSD data set (the same as in [12]), since some terms[1] have meanings, here represented as Concept Unique Identifiers (CUIs), that we were not able to extract a textual definition from the UMLS 2012 database. The knowledge-based results are considered without using these terms.

3 Knowledge-based Approach

3.1 Word Embeddings

We calculated the word embeddings from around 15 million MEDLINE abstracts. The continuous-bag-of-words model architecture [6] was used to generate the word embedding models using the Gensim framework [15] implemented in Python. These generated word embeddings were used to calculate embedding vectors for the CUI textual definitions, and for the surrounding context of each ambiguous term.

3.2 Word Embedding Averaging Functions

In the first place, Inverse Document Frequency (IDF) values were calculated using the CUI textual definitions, where each definition represented a document. We used the IDF formula that is expressed in (1), where N is the total number of documents, and df_t is the document frequency of the term t.

$$\mathrm{IDF}(t) = \log_{10} \frac{N}{df_t} \tag{1}$$

Afterwards, the embedding vectors of the CUI textual definitions were calculated using the Term Frequency – Inverse Document Frequency (TF-IDF) weighting scheme. And on the other hand, the embedding vectors of the surrounding contexts of the ambiguous terms were weighted using the pre-calculated IDF values and a word distance function $\mathrm{f}(d)$.

Distinct word embedding averaging functions were defined for being applied only in the abstracts containing the ambiguous terms. For each word w in the surrounding context of an ambiguous term t, we used its absolute word distance $\mathrm{d}(w, t)$ to calculate a weighted context vector. The absolute word distance $\mathrm{d}(w, t)$

[1] Terms not used: Ca; CNS; Crown; DBA; FAS; Gamma-Interferon; Hybridization; ITP; PCP; Plaque; Pneumocystis; Semen.

is used as input parameter for a specific decay function $\mathrm{f}(d)$. The objective was to give a greater importance to words closest to the ambiguous term. The weighted calculus of the context embedding vector is shown in (2), where the function $\mathrm{WE}(w)$ represents the embedding vector of a specific word w. All the calculated embedding vectors for the CUIs and the contexts were normalized.

$$\text{embedding\_vector}(context) = \sum_{w \in context} \mathrm{IDF}(w) \cdot \mathrm{f}(\mathrm{d}(w,t)) \cdot \mathrm{WE}(w) \qquad (2)$$

Four decay functions were defined and tested for weighting the calculus of the context embedding vectors:

- No decay: $\mathrm{f}(d) = 1$;
- Fractional decay: $\mathrm{f}(d) = 1/d$;
- Exponential decay: $\mathrm{f}(d) = \exp(-d)$;
- Logarithmic decay: $\mathrm{f}(d) = 1/\ln(1+d)$.

3.3 Method

Firstly, CUI textual definitions were extracted from the UMLS database, and each CUI was represented as an embedding vector calculated as a weighted average of the word embeddings with the TF-IDF scheme. Each abstract of the MSH WSD data set, that is, each surrounding context of the ambiguous terms were also mapped to an embedding vector using different word embedding averaging functions as described before.

For each abstract, we could calculate the Cosine Similarity (CS) between the embedding vector of the surrounding context and the embedding vector of each possible meaning (specified by a CUI) to select the CUI with highest cosine similarity as the correct meaning. However, we extended this baseline approach calculating cosine similarities between the context embedding vector and each of the CUI definition vectors. And for weighting these cosine similarities we used CUI-CUI association values that were calculated as normalized Pointwise Mutual Information (nPMI) values from the co-ocurrence of MeSH terms in MEDLINE citations[2]. The score for each possible CUI of an ambiguous term is then calculated as shown in (3), where the CUI that achieves the highest score is inferred as the correct sense.

$$\mathrm{score}(CUI) = \frac{1}{N} \sum_{j} \mathrm{nPMI}(CUI, CUI_j) \cdot \mathrm{CS}(\boldsymbol{t}, \boldsymbol{CUI_j}) \qquad (3)$$

According to (3), for each possible CUI the cosine similarities between its context embedding vector $\boldsymbol{t}$ and the concept embedding vectors $\boldsymbol{CUI_j}$ are weighted by the respective nPMI values. The nPMI value of a CUI in relation to himself has a value of a unit. However, some nPMI association values are undefined,

[2] https://ii.nlm.nih.gov/MRCOC.shtml.

since the nPMI values were restricted using different thresholds. The final division of the score by N is a normalization, since N is the total number of nPMI associations used to weight the N cosine similarities.

4 Results

Simulations with different nPMI thresholds $(0.2, 0.3, \ldots, 1.0)$ were performed. Table 1a shows the results for nPMI = 1, that is the case when only the cosine similarity between the context vector and the CUI vector is considered. On the other hand, table 1b shows the results for nPMI ≥ 0.3, which was the nPMI threshold that produced the best results. The first row of the table 1b shows the results with no decay function, which produced an accuracy of 86% with the word embedding model of size 100 and window of 50 words. From table 1b one can see that the exponential decay weighting obtained the lowest results even when compared to using no decay. The fractional decay produced the best results, achieving an accuracy of 85% with nPMI = 1, and a best accuracy of 88% with nPMI ≥ 0.3. Overall, the use of the concept association values nPMI ≥ 0.3 allowed to improve the overall accuracies around 3% when compared to not using any related concepts, that is, the case when only the cosine similarity between the CUI definition and the context of the ambiguous concept was considered.

Table 1. Knowledge-based accuracies using four different word embedding averaging functions with IDF weighting. f(d): word embedding averaging function, where d is the absolute distance to the ambiguous term; S: size; W: window; nPMI: normalized Pointwise Mutual Information.

(a) nPMI = 1: only the cosine similarity between the context embedding vector and the concept definition vector is used.

f(d)	S100			S300		
	W5	W20	W50	W5	W20	W50
1	0.8078	0.8194	0.8200	0.8077	0.8194	0.8182
$1/d$	0.8375	0.8468	0.8461	0.8408	**0.8471**	0.8456
$\exp(-d)$	0.8227	0.8255	0.8233	0.8262	0.8257	0.8214
$1/\ln(1+d)$	0.8216	0.8347	0.8361	0.8232	0.8344	0.8343

(b) nPMI ≥ 0.3: related concepts with a nPMI value higher than 0.3 are the ones considered to weight the cosine similarities.

f(d)	S100			S300		
	W5	W20	W50	W5	W20	W50
1	0.8430	0.8585	0.8604	0.8428	0.8552	0.8541
$1/d$	0.8638	0.8766	**0.8795**	0.8641	0.8751	0.8747
$\exp(-d)$	0.8418	0.8522	0.8515	0.8421	0.8505	0.8487
$1/\ln(1+d)$	0.8539	0.8686	0.8717	0.8539	0.8674	0.8678

5 Conclusions

We showed that adding a word distance weighting in the calculus of the context embedding vector improved the accuracy of our proposed KB method in 2%, achieving a best accuracy around 88% using the fractional decay. This result is slightly above the results obtained with the three KB methods proposed by [9] which achieved a top accuracy around 84%. Tulkens et al. [16] also applied a similar KB method to the same data set using BioASQ word embeddings, obtaining a disambiguation accuracy of 84%. They also compared UMLS definitions with the contexts of the ambiguous terms. Sabbir et al. [11] used a KB approach with neural concept embeddings and distant supervision, achieving a top state-of-the-art accuracy around 92%.

One restraint of our method is that because we were not able to extract definitions for some CUIs, the disambiguation could not be performed to all terms of the data set. As future work, we intend to overcome this problem searching for textual definitions in other databases. Also, we intend to apply this word embedding distance weighting in a supervised learning approach to verify the veracity of this method.

Acknowledgements. This work was supported by Portuguese national funds through FCT - Foundation for Science and Technology, in the context of the project IF/01694/2013/CP1162/CT0018. Sérgio Matos is funded under the FCT Investigator Programme.

References

1. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticæ Investigationes **30**(1) (2007) 3–26
2. Navigli, R.: Word sense disambiguation: a survey. ACM Computing Surveys **41**(2) (2009) 10:1–10:69
3. McInnes, B.T., Stevenson, M.: Determining the difficulty of word sense disambiguation. Journal of Biomedical Informatics **47** (2014) 83–90
4. Garla, V.N., Brandt, C.: Knowledge-based biomedical word sense disambiguation: an evaluation and application to clinical document classification. Journal of the American Medical Informatics Association **20**(5) (2013) 882
5. Tsai, C.T., Roth, D.: Concept grounding to multiple knowledge bases via indirect supervision. Transactions of the Association for Computational Linguistics **4** (2016) 141–154
6. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv e-print (2013)
7. Wu, Y., Xu, J., Zhang, Y., Xu, H.: Clinical abbreviation disambiguation using neural word embeddings. In: Proceedings of the 2015 Workshop on Biomedical Natural Language Processing (BioNLP 2015), Beijing, China, Association for Computational Linguistics (2015) 171–176
8. Taghipour, K., Ng, H.T.: Semi-supervised word sense disambiguation using word embeddings in general and specific domains. In: Proceedings of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2015), Denver, Colorado, USA (2015) 314–323

Can the Wisdom of the Crowd Be Used to Improve the Creation of Gold-standard for Text Mining applications?

Luis F. Campos, Andre Lamurias, Francisco M. Couto

LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Abstract. Natural Language Processing (NLP) and text mining techniques require annotated datasets to develop and evaluate new approaches. These datasets are commonly developed by domain experts, who manually annotate a corpus of documents with the relevant information. This process is expensive and time-consuming, and for this reason, it has been suggested that crowdsourcing techniques could be used to develop a gold standard, with similar quality.
This paper presents a workflow for using the wisdom of the crowd to develop an annotated corpus. Our approach is focused on the annotation of named entities in biomedical text, using a crowd of non-experts. This workflow is adaptable to various types of text and domains. We tested this workflow on two occasions, one to compare the crowd annotations with a gold standard created by experts and another to annotate clinical reports with radiology terms.

Keywords: wisdom of the crowd; text mining; named-entity recognition

1 Introduction

Natural Language Processing (NLP) and text mining techniques require annotated datasets to develop and evaluate new approaches. These datasets consist of a corpus of documents relevant to a specific domain and its annotations, which can serve as a gold standard to train a machine learning classifier and evaluate its performance. These annotations are made by domain experts and require a defined set of annotation guidelines and time to read and annotate the texts. Recent crowdsourcing techniques have shown that non-experts can provide reliable annotations for text mining tasks. Snow et. al. [1] compared expert annotations with non-expert annotations and obtained a high agreement between the two on 5 different NLP tasks. Furthermore, they have shown that only a small number of non-expert annotators is necessary to achieve expert quality. Li et. al. [2] used crowdsourcing to annotate documents with relations between chemical compounds and diseases. Each chemical-disease pair was reviewed by 5 annotators. Their approach achieved an F-score of 0.505, while machine learning systems trained on a gold standard annotated by experts obtained an F-score of 0.570.

Our objective was to develop a flexible crowdsourcing workflow for annotation of biomedical concepts in text. To demonstrate the flexibility and feasibility of the workflow, we performed two experiments consisting of crowd annotation sessions on two different domains (phenotypes and radiology terms). With one of the sessions, we compared the quality of the crowd annotations with automatic annotations, with the other we applied the workflow to obtain a new annotated dataset. For our purposes, we define crowd as any number of people performing the same task with a common objective. The number of people necessary depends on the amount of text to be annotated in order to reach an acceptable level of quality. Both sessions were organized in the context of a classroom, i.e., the annotation session occurred in the same space during a fixed period of time. This way, we could explain clearly the motivation and objectives of the experiment and solve any technical issues occurring during the annotation process. This paper describes the crowdsourcing workflow we developed for annotation of biomedical documents, as well as the specifics of each annotation session. Furthermore, we present the results obtained with each experiment, a discussion about the advantages of this approach and further improvements.

2 Methods

In this section, we will first explain in general terms the idea of using the wisdom of the crowd to develop a gold standard and then focus on the two experiences we had implementing it. A crowd annotation session consists of a period of time where human annotators are asked to annotate documents according to some guidelines. Our goal with a crowd annotation session is to get annotations of terms of a certain domain on a set of documents identified by the session participants. To accomplish this goal, it is necessary to carefully prepare each annotation session.

The first step should be to define the domain and a respective set of documents relevant to that domain. The source of these documents will depend on the nature of the experiment: if the objective is to evaluate crowd annotations on a specific domain, a corpus with gold standard annotations should be chosen, so that the crowd annotations can be evaluated against the gold standard. However, if the objective is to develop a new gold standard, or to improve automatic annotations, it is necessary to select an appropriate set of documents to annotate. There are several source of biomedical text that can be used to this end, such as PubMed and ClinicalTrials.gov. The number of documents should be chosen according to duration of the sessions. Since the participants are not domain experts, they will have to spend time reading the guidelines and verifying the definition of various concepts.

After defining the topic and corpus, it is necessary to define the type of information to be annotated. In our case, we focused on defining named entity annotations, which consists of identifying the words that refer to entities of a specific domain. It should be clear to the participants what should be annotated in the text. Hence, an existing vocabulary or ontology can be used, so that the

annotators have plenty examples of the entities. In our experiments, we used two ontologies that provide a web interface that can be used to search for terms, and most terms contain a definition. This way, the annotators can confirm if an entity should be annotated or not based on its definition. Other criteria should also be defined, similar to the annotation guidelines used for other corpus annotation projects. Krallinger et. al. [3] provides a detailed description of the guidelines used to annotated chemical compounds by experts. In our case, the guidelines should be simple to understand by non-experts. As such, we compiled a list bullet points containing the main principles that the annotators could consult during the session.

It is necessary to choose how the annotators can access the text and submit the annotations. There are various frameworks that can be used to this purpose. Commercial frameworks, such as CrowdFlower and Amazon Mechanical Turk, provide more options and features and can recruit annotators from a large pool of users, which are paid for each annotation or document. However, there are other examples of volunteer-based crowdsourcing projects, such as Folding@Home [4] and Mark2Cure [5]. We used WebAnno [6] for the two experiments because it was freely available and adaptable to our needs; i.e. it was possible to upload documents and organize them in projects, and the annotators could access the documents and directly annotate each document. Then, we could export the annotations of each user and analyze them with our scripts. During the sessions, we demonstrate to the users the annotation guidelines and how they should use the annotation platform and the guidelines for doing it. To bootstrap the annotation process we automatically pre-annotate the corpus with terms of the domain, based on an existing vocabulary. This way, the annotators can accept and reject the automatic annotations, or add new ones.

2.1 Evaluation metrics

After each annotation session, we show and discuss the results with the participants. We chose to analyze the results right after the end of the session and share with the participants, to emphasize the importance of their work. For the HPO experiment, for which we had a gold standard, we calculated precision, recall, and F1-score. For both experiments, we compared the annotations of the participants based on an agreement score. The agreement score measures how agreeable is each participant, comparing the annotations of the majority of the participants. We wanted to obtain a high score to the participants that voted most times like the majority, and a low score to participants that voted differently more often. By ranking the participants by this agreement score, we could filter out noisy annotations. We defined the agreement score of a participant as the sum of the fraction of users who agreed with the participant on each annotation. Since we provided automatically generated annotations for each document, each participant could either accept or reject each annotation:

$$ballot_{a,p} = \begin{cases} 1 & \text{if participant } p \text{ validated annotation } a \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where p is a participant and a an automatic annotation. We can compare the ballot of two users on the same annotation:

$$v(a, p1, p2) = \begin{cases} 1 & \text{if } ballot_{a,p1} = ballot_{a,p2} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Hence, we calculated the agreement score with the following formula:

$$agreement\_score(p) = \sum_{a \in A} \frac{1}{\#P} \sum_{i \in P} v(a, p, i) \tag{3}$$

where A is the set of automatic annotations and P is the set of all participants. For example, considering an annotation accepted by 70% of the participants, means that those who accepted receive a score of 0.7 for that annotation, while the ones who rejected it would receive score of 0.3. Hence, this score is related to how much the annotations made by the participant have in common with the crowd.

The agreement score only takes into account how the crowd performed on pre-existing annotations, even though they could also add new annotations. We decided to not include new annotations in the agreement score since it is a different task from simply reviewing automatic annotations, comparing to searching actively for missing annotations. We assigned a novelty score to each participant, which measures the number of novel annotations added by each user:

$$novelty\_score(p, t) = \sum_{a \in A} ballot(p, a) \text{ if } \frac{1}{\#P} \sum_{i \in P} ballot(i, a) \geq t \tag{4}$$

where A is the set of annotations added by the participants. We can change the threshold t whether we want to reward novel annotations regardless if they were also added by other participants. The agreement and novelty scores can be used to rank the participants of the annotation session and study the differences within a crowd of annotators.

2.2 HPO terms annotation

In this annotation session, the participants had to annotate a specific part of a corpus [7] with HPO (Human Phenotype Ontology) [8] terms. The participants of this session were a group of 13 students enrolled in a systems biology PhD program. The gold standard annotations of this corpus allowed us to compare the crowd annotations with the gold standard. We used MER [9] to obtain baseline annotations which participants could accept or reject, or also add new annotations. Six documents from this corpus were annotated during approximately 40 minutes. Material related to this session can be found at https://github.com/lasigeBioTM/HPO_Crowd_Annotation_Experiment.

2.3 Radiology terms annotation

In the context of a biomedical text mining workshop, we organized hands-on activity consisting of the annotation of RadLex terms [10] on Radiology reports by the workshop participants. We selected three Radiology reports from Lurie Children's TF Library http://mirc.luriechildrens.org/query. We performed an informal preliminary test to understand how long it would take to annotate each report and other issues that could occur. Since we had limited time to perform this activity, we decided to select 3 documents for this session, expecting an average of 10 minutes per document. We also used the NCBO annotator [11] to present automatic annotations to the participants. The rationale for this activity was that the aggregate annotations of the participants would be used as a wisdom of the crowd gold standard, which could be used to train and evaluate machine learning approaches. Material related to this session can be found at https://github.com/lasigeBioTM/biomedical_workshop_bod.

3 Results and Discussion

3.1 HPO annotation session

During the HPO annotation session, we obtained annotations of six documents from 13 participants. The crowd annotations were created as described in the previous section and compared with the gold standard annotations developed by domain experts. Here the rationale was to check if the wisdom of the crowd could replace the use of (expensive) experts in the development of gold standard annotations. We experimented with a number of validation thresholds to study which one works best. This validation threshold corresponds to the fraction of participants that have to agree on an annotation to accept it.

Using different validation thresholds, we observe that the precision values tends to increase with the threshold, while the opposite happens to the recall (see Table 3.1 and Figure 3.1). The maximum F1-score is obtained with a threshold of 0.6 (0.659) while the maximum precision is obtained with a threshold of 0.7 (0.815) and maximum recall with a threshold of 0.1 (0.868). The plot in Figure 3.1 is similar to the analogous ones presented in [12, 5], which describe similar experiments, but on a larger scale (more participants and using more documents) and in a different domain (disease mentions). These differences might explain why in our case we obtained worse F1-Scores.

The F1-score obtained by our baseline (MER annotations) on the same 6 documents was 0.618 (Table 3.1. Using the crowd with a threshold of 0.6, we obtained a higher F1-score (0.659). These results show that the crowdsourcing approach outperformed a rule-based system, especially in terms of precision. We have previously developed IHP [1], a system based on machine learning and post-processing rules to recognize human phenotypes. This system was evaluated with cross-validation on the full HPO corpus. We considered only the same 6

[1] https://github.com/lasigeBioTM/IHP

Threshold	Precision	Recall	F1-Score
0.1	0.315	**0.868**	0.462
0.2	0.429	0.849	0.462
0.3	0.500	0.774	0.607
0.4	0.547	0.660	0.598
0.5	0.660	0.623	0.641
0.6	0.789	0.566	**0.659**
0.7	**0.815**	0.415	0.550
0.8	0.800	0.302	0.438
0.9	0.786	0.201	0.328

Table 1. Metrics evaluating the quality of the crowd annotations against gold standard annotation, using different aggregation thresholds, e.g., the 0.5 row means the only annotations accepted by at least 50% of the participants were included in the crowd annotations.

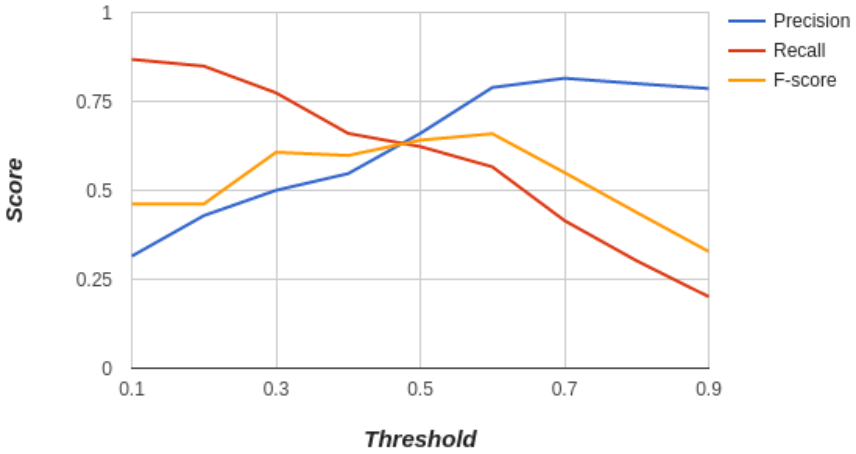


Fig. 1. Graphical representation of the scores obtained with various validation thresholds.

documents and calculated the same scores, to compare with the scores obtained by the crowd. While the crowd obtained higher precision, IHP obtained a higher recall, resulting in a higher F1-score. However, it should be taken into account that IHP was highly tuned for this corpus, using the gold standard annotations and manually tuned post-processing rules, while the crowd annotations were obtained in approximately 40 minutes. Furthermore, a larger number of documents may be necessary to perform a stronger comparison.

	Precision	Recall	F1-Score
MER	0.596	0.642	0.618
Crowd	**0.789**	0.566	0.659
IHP	0.577	**0.849**	**0.687**

Table 2. Comparison between a baseline approach (MER), crowd annotations, using the highest F-score threshold (0.6), and machine learning and post-processing rules (IHP).

3.2 RadLex annotation session

The RadLex annotation session resulted in 3 radiology reports annotated by 25 participants. We created a crowd annotation file for each of the radiology reports being used by including in the crowd annotation file just the annotations that were done by at least 80% of session participants. Since we did not have expert gold standard annotations for this corpus, we analyzed manually the effect of the crowd on the baseline annotations. We observed that the crowd was able to improve the specificity of the annotations in some cases. While the baseline annotations contained "left retinoblastoma" as separate entities, most of the participants merged the two entities. In other cases, the crowd added more words to an entity, changing "tumor" to "pineal tumor", for example.

Another case that demonstrates the potential of the crowd was the identification of entities with typos. In one of the reports, the word "adolsecent" is used, and it is clear that the author meant "adolescent", a radiology term according to RadLex. Automatic approaches to detect typos are limited in terms of performance, and vocabularies to do not include this type of variation. The fact that the crowd was able to detect this issue is another advantage of this approach.

3.3 Comparison between annotation sessions

We calculated the agreement score for each participant of the two annotation sessions and normalized the value by dividing by the total number of annotations. The maximum agreement score obtained during the HPO session was 0.730 while for the RadLex session it was 0.832. While the radiology reports may be more difficult to understand for non-experts, the HPO vocabulary seemed to be more ambiguous. During the HPO session the participants had more doubts about what was relevant, resulting in more disagreements in the annotations. Comparing the ranking of participants, we noticed that some participants obtained both high agreement and novelty scores. In the HPO session, the top 5 of both rankings had 3 participants in common, while in the RadLex session had, there were 2 participants in common. We also identified one participant in the RadLex session who was in the top 5 of the agreement score ranking and in the bottom 5 of the novelty score. This case can have various explanations: the participant could have had technical difficulties in adding new annotations, could

have been more focused on validating the automatic annotations, or simply did not think that many annotations were missing.

4 Conclusion

In this paper, we present our workflow for crowdsourcing annotations using non-experts. We demonstrated this workflow on two sessions: one where a gold standard was available and it was possible to evaluate the crowd annotations, and another where no gold standard was available. With the first dataset we obtained an F1-score superior to a dictionary matching approach and using different validation thresholds, we can obtain higher precision or recall values. With the second dataset, we demonstrated how the workflow can be applied to different domains and types of text. In this case, the documents consisted of clinical reports, and the crowd was able to improve the quality of the baseline annotations.

The experiments presented in this paper represent a proof-of-concept for future, more extensive, crowd annotation projects. While in both experiments every participant annotated every document, some authors suggest that only a few annotators are necessary to annotate a document [2]. This way, using the same number of participants and in the same time duration, we could obtain a larger number of annotated documents. More complex text mining tasks can also be performed using the wisdom of the crowd [13]. Named entity normalization and relation extraction are two tasks that can be attempted in the future using our workflow [14, 15]. This workflow can also be integrated with active learning algorithms to select the best annotators for each document [16].

Acknowledgments. This work was supported by the Fundação para a Ciência e a Tecnologia (https://www.fct.mctes.pt/) through the PhD Grant ref. PD/BD/106083/2015 and UID/CEC/00408/2013 (LaSIGE).

References

1. Snow, R., Connor, B.O., Jurafsky, D., Ng, A.Y., Labs, D., St, C.: Cheap and fast - but is it good? Evaluation non-expert annotiations for natural language tasks. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) (October) (2008) 254–263
2. Li, T.S., Bravo, A., Furlong, L.I., Good, B.M., Su, A.I.: A crowdsourcing workflow for extracting chemical-induced disease relations from free text. Database (Oxford) (2016) 1–11
3. Krallinger, M., Rabal, O., Leitner, F., Vazquez, M., Salgado, D., Lu, Z., Leaman, R., Lu, Y., Ji, D., Lowe, D.M., Sayle, R.A., Batista-Navarro, R.T., Rak, R., Huber, T., Rocktschel, T., Matos, S., Campos, D., Tang, B., Xu, H., Munkhdalai, T., Ryu, K.H., Ramanan, S.V., Nathan, S., Zitnik, S., Bajec, M., Weber, L., Irmer, M., Akhondi, S.A., Kors, J.A., Xu, S., An, X., Sikdar, U.K., Ekbal, A., Yoshioka, M., Dieb, T.M., Choi, M., Verspoor, K., Khabsa, M., Giles, C.L., Liu, H., Ravikumar, K.E., Lamurias, A., Couto, F.M., Dai, H.J., Tsai, R.T.H., Ata, C., Can, T., Usi,

IoT Solutions for Traffic Characterization in Smart Cities

Daniel Silva, João Paulo Barraca, and Diogo Gomes

Institute of Telecommunications - Aveiro

Abstract. Agencies managing Road traffic need to make informed decisions, in order to define which road sections have the greatest risk of traffic-related impacts. In this context, it is recognized that the implementation of Advanced Traffic Management System (ATMS) may not only improve network efficiency, but also minimize other traffic externalities. In this context, novel software and hardware solutions, capable of providing improved information from the traffic dynamics, can play an important role in the knowledge we have, and the way ATMS is executed. In particular, the availability of geo-referenced data is increasing quickly, either from nomadic devices as well as from social media, and monitoring sensors networks. One of the challenges is to combine and to increase the potential of each source of information, and then combine multiple sources together in an aggregate model. In this paper we describe the architecture and implementation of an accurate, high frequency vehicle tracker, with integration of real time engine statistics, and enhanced with an autonomous inertial model. This tracker is able to acquire fine details of a driver's behavior, and through the interconnection with an IoT platform provide data in standardized formats. We also describe the accompanying platform, analysis and visualization modules, that are part of a novel ATMS.

Keywords: ATMS, road traffic, vehicle tracker, floating car data, driver's behavior

1 Introduction

The increase in road traffic leads to high levels of stress in drivers, in pollution, as well as management complexity. As such, there has been an increasing interest in developing intelligent ways to help drivers cope with today's traffic, and to increase the knowledge we have from the infrastructure. Crowdsourcing sources, usually named as sources of floating car data, can gather data about drivers, pedestrians and other entities, resorting to Internet of Things (IoT) platforms to manage the flow of data. By combining analytical tools to process the information, it is possible to notify users, so they can take advantage of safer roads, better environment and adopt more sustainable choices. Entities responsible for managing roads, and related infrastructures are able to have information with higher relevancy, and take decisions of greater impact.

Nowadays, the road traffic is getting more stressful, and if we go to bigger cities it is even worst. The traffic congestion can increase and decline for several reasons. It might be caused by the rush hour in some times of the day or by other events such as

road management, accidents os police checkpoints as explained in [23]. Congestion is not heterogeneously spread in the city but it concentrates in some congestion points. These points represent a threat to the drivers which are exposed to pollution coming from exhausts [19].

To understand the troubled points in a city, like roads with too much daily traffic concentrated or holes in the pavement, which in some cases results in slowdown of the traffic, with accidents or flat tires, we must have some way of knowing where those points are situated in the city. Identifying those points requires a decentralized method to collect data and somehow cross all data to get a useful conclusion. Knowing, with precision, how drivers behave in these situations is of paramount importance. This work presents a first step in that direction.

In section 2 we present a set of solutions that are relevant to our scenario. Section 3 presents and overview of the Advanced Traffic Management System (ATMS) solution created. Section 4 complements this with the description of the trackers created. Section 5 addressed the evaluation of both the platform and tracker. Finally, in section 6 we draw some conclusions regarding this work, and the future directions to follow.

2 Related solutions

For the specific scenario of traffic monitoring, data can be gathered using two distinct methodologies: through dedicated sensors, or using methods based on crowdsourcing. Dedicated sensors can be inside the car, or they can reside in the road infrastructure. A common example is ZELT, which consists of an Inductive Loop Detector (ILD) able to count vehicles with great precision. Useful for simple managing of a car park, where it counts the number of cars inside the park and informs the drivers if the park is either full or not, with an accuracy of +/-5%. There are three fundamental traffic parameters able to be measured with ILD such as speed, counting and capacity, obtained with a single or double ILDs (for speed detection) [15]. This system will work with all kind of cars, because they have metal in the wheels, but some vehicles, like two-wheel motorized, might be omitted on the count.

Attending to traditional solutions based on CCTV cameras, monitoring traffic is an obsolete technique, and is unreliable when we talk about hundreds of vehicles [2]. The IoT comes here to help and solve this problem. In one such example, [2] introduces a new concept of traffic monitoring. They create the Internet of Vehicles (IoV), and then connect some existent elements to nowadays roads, like traffic lights, CCTV cameras and speed trackers, for the purpose of enabling the possibility to take decisions via pre-embedded algorithms [2].

Satellite based sensors work with the in-car system reading the car's position repeatedly using the Global Positioning System (GPS). Then the position is used to determine the vehicle's speed and the road where the vehicle is positioned. The system knows what speeds are practiced for each road, under non-congestion conditions, and thus, expects a determined speed to be reached by the vehicle. This information, has to be accurate, reliable, timely and complete. So, the traffic information is collected with ILDs, like the one mentioned before, and also with video cameras. This methods, only provides

information from the place where they are installed or in their coverage area which lead us to a need of more video cameras to cover bigger areas, and more investment on ILDs.

Therefore it's possible to automatically detect traffic congestion, analyzing when the vehicle's speed is much lower than the expected speed, with the possibility to refine the algorithm used for traffic detection [10]. Other solutions explore these or similar principles, and are basically able to count vehicles, or determining average speed with low precision.

Methods based on crowdsourcing rely on dividing the monitoring needs by a group of people, typically the drivers itself. Each element gathers localized information that can then be processed to obtain a global view. An example are data loggers, nowadays used by insurance companies, which are not more than a device that acquires and stores sensor data internally. The data can be gathered from multiple sensors like GPS and accelerometer, and a smartphone can act as a mobile data logger. However most of instrument manufacturers consider a data logger to be a stand alone device. The great problem with data loggers is the high price, when most of the times people with smartphones can collect similar data.

A promising application of these techniques has been applied on traffic monitoring, by controlling and optimizing the routes that are recommended for the users to reduce fuel consumption and carbon emission on traffic jams [6].

The presence of smartphones in cars, provides a way to implement sensor networks and driver assistance systems, as well as other Intelligent Transportation System (ITS) applications [3]. It is possible to detect road anomalies by analyzing driver behaviors. Analyzing smartphone inertial sensors to calculate the angle swerving and gyroscope drift, can become reliably to detect swerve [18]. There is a considerable amount of research regarding driver behavior analysis based on mobile phones [7, 16, 21].

Data gathered from crowdsource can be analyzed using data mining techniques allowing us to learn relevant/hidden pattern. These patterns can then be used to optimize traffic flow, prevent road accidents, prioritize road repair amongst other possibilities.

Crowdsource information can also be used to assess driving risks, allowing drivers to change their behavior and city halls to proactively develop safety countermeasures [5, 12, 14] [8].

Finally, crowdsource information can be used to identify traffic jams [4], locations prone to road accidents [11, 13] and more important be used to plan, develop and implement better road structures [9, 22]. This is particularly true when data is gathered into a data centric platform, which allows further processing and correlation, a good example is the IRIS software stack [20]. However, ATMS systems are frequently limited in scope, as they are focused in data aggregates with low accuracy, and related to the movement of a reasonable amount of vehicles, failing to handle per vehicle behavior as we aim.

3 Platform for an ATMS

The idea of collecting data from different sensors and combine them to have some kind of characterization of the traffic, presents serious challenges. In part because sometimes it is not known what data can be useful, or what frequency can be used. More

than often, data becomes meaningful after analysis and otherwise irrelevant sources of information become vital. As a simple example, it should be considered that the driving behavior changes during the day of the week, the time of the day, the weather conditions, or even due to the music that it is being played in the radio. While we cannot observe all aspects contributing to the behavior of a driver (as some as intrinsic to the driver), our approach is to acquire as many sources as possible, and then allow data to be correlated and analyzed.

The proposed solution was initially focused in the creation of a mobile application, capable of acquiring floating car data and distributed to a wide audience. Then, as shown in Figure 1, an in-car tracker, with higher acquisition rate was developed. Both are described in the next section. Also, we aimed at creating a platform to integrate other sources of information that could potentially be used to determine driving behavior, as well as other impacts of an aggregate number of drivers. At this moment we integrate localized, and very detailed, weather information, and later we plan to integrate air monitoring sensors that were previously deployed in our city.

There are 4 distinct modules. The floating car data, as a mobile application to provide an easy and scalable data source using crowdsourcing. Then a car tracker as the main source of data, more reliable and with a faster sampling rate. This last component publishes all the data to the IoT Platform which we explain further ahead in this section. The backend is where the logic of the collected data happens. This means that the backend stores trips and devices which will be provided then to the last module, the dashboard, where the user will have the possibility to analyze the different metrics collected.

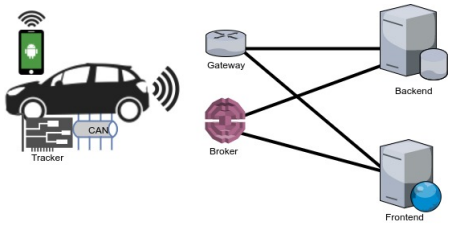


Fig. 1. System architecture

The core of our platform consists of an IoT driven platform, and we followed the best practices regarding a scalable and redundant, event driven platform for handling large amounts of data and multiple sources and it is possible to make queries to the database. We resort to a message broker and the Message Queuing Telemetry Transport (MQTT) protocol [1]. The platform consists of several components such as a gateway to receive data, a broker, several data sources providing information from external services, a database, and then an applicational backend and frontend components, implement the ATMS logic and allow users to interact with the system. This technology does not rely on the type of the internet connection, so it can be through mobile networks like 3G or 4G or other such as Wi-Fi and Ethernet.

On top of this platform we developed a frontend dashboard, to administer all existing devices and users, and of course explore the data collected. This platform will allow analyzing the collected data in multiple dimension, and currently already allows visualizing data in simple, yet detailed charts.

Our approach was to organize data based on the concept of trips. Each device when initiated, automatically creates a new trip, and this is exposed in the ATMS platform. Afterwards, the device will start collecting data and publish data which can be correlated with other information from the existing sources. The complete life-cycle of a trip is based on 4 steps: (i) Init; (ii) Start Trip; (iii) Collect and Publish; (iv) Stop Trip; and then start over again.

After the devices make at least one trip, data will be available in the dashboard for analysis.

3.1 Backend

The backend is one of the central modules of our solution and was developed using the Django Framework version 1.10. The Django Framework let us place many system components in one place only, instead of placing them separately. So, we can have a REST API to handle all needed requests, and also have a local database to store everything required, like users and devices in existence. Besides all this, we can serve the frontend from the same system, which will be detailed further in section 3.2.

The API was developed to handle login requests, as experimental data is bound to specific privacy agreements, and requires users or systems to have valid credentials. With this API, and the correct set of authentication data, it is possible to obtain the necessary interaction between the devices, the storage and with the dashboard.

The user data model is based on a custom user model, created to complement the default Django user model. We required new fields such, as the phone and most importantly the role. The role field gives us a very important detail to separate hierarchically the different users. We do not want to give the same permissions to everyone, so the default user permissions will have restricted access to the website. Then the staff and administrator roles have more permissions.

Devices are mainly identified by an Universal Unique Identifier (UUID). We designed a specific data model for devices and the principal characteristics are the owner, the UUID, the type and its internal name. Based on these elements, we can attribute a unique device for each smartphone or in-car unit. The data model for this represents all devices in the system and has detailed information about each device.

A necessary model we designed was the one which maps the sensors of each device. In other words, each device has its specific sensors like the accelerometer, gyroscope, speed, GPS, revolutions per minute, and the others. As the devices are not all homogeneous, we use this model to specify the sensors present in each device. For instance, a smartphone doesn't have anything related to the engine metrics. Thus, this model was developed to simplify and to become more dynamic for future additional types of devices and more sensors.

One of the key data models for this solution was the one related to trips. Specifically we aimed to simplify the interpretation of what is collected from all devices, as data

sources can be massive. The objective was to divide device usage in trips, as this way we can have smaller pieces of data to analyze.

3.2 Dashboard

The purpose of the dashboard is to provide users with the means to interact with the system and explore the data stored. It was developed around the Django Framework, similarly to the backend, but now also integrating technologies such as Angular JS, which makes the websites extraordinarily expressive, readable, responsive, and quick to develop.

The MQTT protocol is also used by this component, but now as a subscriber of information. In particular the client subscribes to topics that were created for this application, and which will contain data relevant to the frontend. This will allow the frontend to access real-time data as it is transmitted to the ATMS platform, either in the form or alerts, other notifications and even live data.

Data is accessed by querying the backend components, and specifically the database. A set of APIs provide all the data needed by the dashboard, either historical or live. When a user wants to examine a trip, the website must get the correspondent data for that specific trip, which should be between the intervals of the start and finish timestamps, defined in the backend for that trip.

The great advantage of this dashboard is to get all data together in one place. The capacity of analyzing with great detail each fragment of the trip is a much welcome functionality by ATMS experts. The user is capable of making an analysis of the driver's behavior by looking at the attitude of the device, and compare it to the location where it was measured.

4 Prototype Trackers

In the scope of this work, both to exercise the ATMS platform and to actually obtain a real world dataset, two different trackers were developed: a mobile tracker based on a smartphone, and an in-car unit. An additional objective of this choice was to create groundwork to evaluate the usefulness of floating car data versus a dedicated in-car unit, which will be presented in future publication.

4.1 Mobile Application

The concept of a mobile application evolved from the need of a simple, yet easily distributable data logger, with a simplified model for updates, and data reporting. With this in mind we developed a custom purpose Android application, able to capture data from the sensors available. We focused on Android due to the larger market reach, which will enable this study to be scaled faster, and thus we adopted a mobile application instead of a data logger.

The mobile application is focused in acquiring data from a group of sensors, through which we can understand the attitude of the smartphone at a given location, and thus extrapolate driving behavior, as well as fuel consumption. The actual number of sensors

acquired varies with each specific smartphone, as we aimed for maximum compatibility, and limited our application to API version 14 and newer. This covers all currently sold Android phones and most phones that are at least 3 years old. If possible, due to the hardware of the smartphone, we will capture the following data: GPS, Accelerometer, Gyroscope, Magnetometer, Barometer. The GPS will provide data every 1 second, while the remaining sensors will provide data at a much higher rate. The accelerometer can be used to determine acceleration profiles (when is the driver braking, how it accelerates, what are the road conditions), the gyroscope allows determining lane changes and slope changes, the magnetometer allows to determine the vehicle direction, while the barometer can also help determining road slopes.

It should be noticed that, Android applications are expected to behave nicely with each other, which means that there are limits in the acquisition speed. In our case the application was developed to read sensors values with a period of 80 milliseconds, which is reasonable fast and, potentially enough for ATMS. Everything that is collected is sent to the platform, with a periodicity of 250 milliseconds. This periodicity was chosen to have in count the battery consumption produced by the application, and thus to have a low battery consumption. If no connection is successfully made the data is stored locally using a database, and later, when a connection is established, this data can be submitted to the ATMS platform using MQTT.

One important metric which is not extracted directly from the sensors but is computed is the attitude. In our case we chose to fuse the raw data from both accelerometer and gyroscope and compute the attitude through a complementary filter [17]. This technique analyses the angle changes reported by the gyroscope over a given period, and correlates those with the accelerometer . With this method we get two angles, one in X axis and the other in the Y axis. These angles will be used to interpret the smartphone attitude over the time, and have a better indication of the driving behavior.

The application gives the user the possibility of running it in the background, as well as in off-line mode. This enables the execution of experiments without network connectivity. If desired, data can also be exported to files with Comma-separated values (CSV).

4.2 In-car Unit

The In-car unit is composed of a Raspberry Pi 3 platform, where we integrated a set of sensors suitable to our scenario, and is shown in Figure 2.

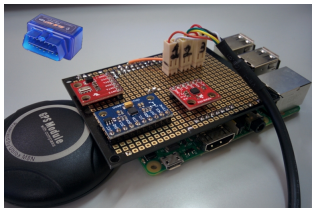


Fig. 2. Prototype of the tracker

In particular we integrated a UBlox M8N-NEO GPS module that allows sampling at 18 Hz, a MMA8452Q Triple Axis Accelerometer, a MPU-9250 3 Axis-Gyroscope (plus 3 Axis Accelerometer), and a MPL3115A2 Pressure and Temperature sensor. The purpose of this unit follows the same principles as the mobile application, but has several distinct features which should be noticed: it allows sampling sensors at a much higher rate, it has a higher precision GPS module. Moreover, because there are less applications running in our system, acquisition jitter is expected to be better. To complement these features, the in-car unit was designed to be used on any vehicle exposing a Controller Area Network (CAN) bus, through an OBDII port. Therefore, this unit can acquire real data that is exposed on the CAN bus, which can be used to have a better insight of the driving characteristics. From the CAN bus is possible to get a large number of parameters such as speed, revolutions per minute, fuel level, fuel pressure, throttle position, coolant temperature between others, but each car has its specifications which presented a big challenge. This means that in practice one can not read all parameters from every vehicle as many are not known. Knowing this, we use only the most common values with relevancy to our work. In particular, the parameters that we acquire are: Fuel Level (percentage), Speed, Revolutions per Minute, Engine Load, Coolant Temperature and Run Time. As an example, the RPM will allow us to determine gear changes, while combined with the Engine Load we can extrapolate the amount of pollutants being emitted. This can be further improved by considering the Coolant Temperature, which detects weather the vehicle is at its optimal performance.

Data from sensors and OBDII are acquired doing oversampling of >1 kilohertz (kHz), and then reported at that frequency, or in fixed periods: e.g. 250 milliseconds (ms). Data is also stored locally and then published to the ATMS platform through an Wifi interface and using the MQTT Protocol.

Similar to the previous case, a complementary filter is applied to the accelerometer and gyroscope in order to determine the actual attitude, which is also reported.

5 Evaluation and results

Our evaluation aims at demonstrating an early prototype of the ATMS we are creating. We can demonstrate the level of detail that can be extracted from a smartphone, or an in-car unit, as well as the capability to visualize specific metrics that are relevant to the determination of the user driving behavior.

The map with the trip path in Figure 3. Here is possible to see with detail, all roads of the trip and other geographic data provided by the Open Street Maps (OSM). We are talking about a precision of less than 1.5 meters. In Figure 3 the zoomed area shows the path of the vehicle exiting from one road and entering the highway. It very perceptible how the driver lead the vehicle, and which transit road was taken.

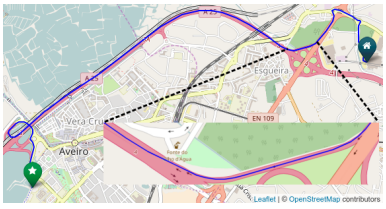


Fig. 3. Map with geo-referenced data from the trip.

Figure 4 depicts the case where the data from a single trips is visualized. In this case we can take a

look at multiple metrics of the same vehicle over a
single trip, and visually we can detect correlations
between them. As an example it is possible to detect that at T=23:43s the car stopped, as determined by the 0 speed, followed by an increased engine load. In the same figure, it can be seen that the green line is directly related with the orange one. The orange line represents the engine load, and it goes up when the driver accelerates, exercising a load into the engine. This metric is expressed as a percentage. We can see that the green line goes up with the orange after a small delay, which is exactly the expected behavior: when the driver accelerates, it's not felt immediately and will always have a delay. Several methodologies can take this information and determine the level of emissions and the fuel consumed.

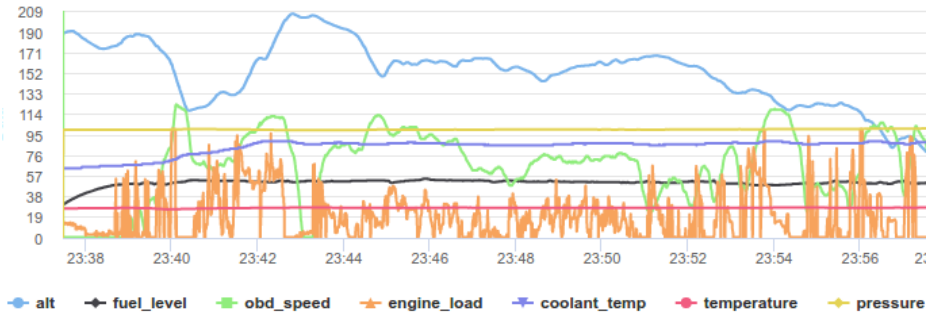


Fig. 4. CAN bus data as obtained by a single vehicle

Automated analysis is expected as our platform is able to support that functionality, but will be postponed to a following work.

In Figure 5 are represented the values obtained for revolutions per minute for a section of the trip. The peaks present in the chart are associated with gear shifts, especially the ones that have a slow rise and then a sudden slow might be representatives of this. Such information can be used to determine how users use their engine, and if applied to different roads, how the road (pedestrians crossing, roundabouts) influence driving behavior.

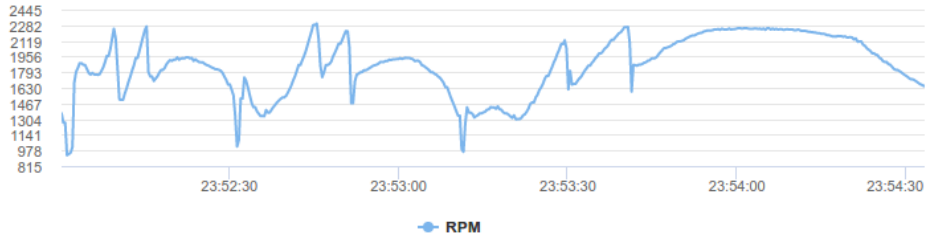


Fig. 5. RPM over time as presented in our platform.

We can also identify when the driver is putting a lower gear, because it produces a wave over the time, without peaks. But to be sure if it is really a gear reduction, usually, the wave comes after one or more peaks of acceleration. In Figure 5, in the last two peaks in the right, it is possible to watch this happening. After the two peaks, the driver makes a gear reduction, and the revolutions of the engine go up slowly, until reaching a point where the inertial movement of the vehicle is equal to zero and then starts to slow down (this behavior is usually referred to as "breaking with the engine").

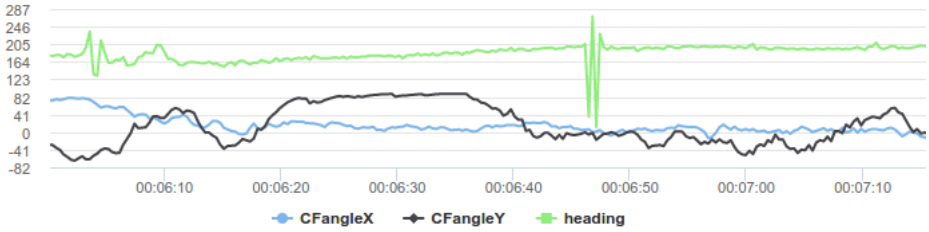


Fig. 6. Produced chart by the dashboard, to analyze the car's attitude over time.

Figure 6 shows the same trip excerpt as the RPM chart, but with different metrics. The blue and the black lines represents the angles for the X axis and Y axis respectively. The green line represents the heading angle of the device. These metrics were computed from the raw data of the accelerometer, gyroscope and magnetometer sensors, and then processed using the complementary filter. By analyzing the data represented by the black line it is possible to determine when the driver makes a curve, either due to the road or when changing lane. As observed in the same chart, heading is not always reliable and sometimes will introduce error. We assume this to be due to magnetic fields existing inside a typical car.

6 Conclusion

This work allows us to learn how to look into the metrics from two distinct types of data collectors and use that information to get an analysis of the behavior of the drivers. The result of this work shows how it can be useful to monitor the driver's behavior, and relate the results with the road conditions and the chosen route by the driver. Moreover, the prototype ATMS architecture, allowed us to collect, store and visualize data from specific trips, across several metrics.

As future work, we want to add a new feature to the system, which includes adding the information about the car brand and model where some device will be used, in order to compute the emissions of Carbon Dioxide (CO2) based on reference values from the manufacturers. With this information, we can compute the Vehicle Specific Power (VSP) parameter, and apply the emission modeling to give us the VSP expressed in kW/Metric Ton for each second. This is an important metric to have in account, since it will provide a new variable in the process of characterizing the driver's behavior, watching the VSP

level. Also we plan to integrate automated processing mechanisms to correlate data in real time, and to explore the differences between floating car data and in-vehicle trackers.

Acknowledgments This work was partially funded by Portuguese FCT grant 9471 – Reforçar a Investigação, o Desenvolvimento Tecnológico e a Inovação (Project 9471 – RIDTI) e partially funded by the European Regional Development Fund (ERDF).

References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials 17(4), 2347–2376 (2015), http://ieeexplore.ieee.org/document/7123563/
2. Dandala, T.T., Krishnamurth, V., Alwan, R.: Internet of Vehicles (10 V) for Traffic Management. In: IEEE International Conference on Computer, Communication and Signal Processing (lCCCSP-2017) (2017), http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7944096
3. Engelbrecht, J., Booysen, M.J., Bruwer, F.J., van Rooyen, G.J.: Survey of smartphone-based sensing in vehicles for intelligent transportation system applications. IET Intelligent Transport Systems 9(10), 924–935 (dec 2015), http://digital-library.theiet.org/content/journals/10.1049/iet-its.2014.0248
4. Gecchele, G., Rossi, R., Gastaldi, M., Caprini, A.: Data Mining Methods for Traffic Monitoring Data Analysis: A case study. Procedia - Social and Behavioral Sciences 20, 455–464 (2011), http://linkinghub.elsevier.com/retrieve/pii/S1877042811014327
5. Guo, F., Fang, Y.: Individual driver risk assessment using naturalistic driving data. Accident Analysis & Prevention 61, 3–9 (2013), http://www.sciencedirect.com/science/article/pii/S0001457512002382
6. Higuchi, T., Yamaguchi, H., Higashino, T.: Mobile Devices as an Infrastructure: A Survey of Opportunistic Sensing Technology. Journal of Information Processing 23(2), 94–104 (2015), https://www.jstage.jst.go.jp/article/ipsjjip/23/2/23{_}94/{_}pdf
7. Johnson, D.A., Trivedi, M.M.: Driving style recognition using a smartphone as a sensor platform. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC). pp. 1609–1615. IEEE (oct 2011), http://ieeexplore.ieee.org/document/6083078/
8. Jonasson, J.K., Rootzén, H.: Internal validation of near-crashes in naturalistic driving studies: A continuous and multivariate approach. Accident Analysis & Prevention 62, 102–109 (jan 2014), http://www.ncbi.nlm.nih.gov/pubmed/24144495
9. Komninos, N., Tsarchopoulos, P., Kakderi, C.: New services design for smart cities: a planning roadmap for user-driven innovation. Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities pp. 29–38 (2014), http://dl.acm.org/citation.cfm?doid=2633661.2633664
10. Kristiansen, E., Claude Loisy, Bosch, W.V.D.: Road Traffic Monitoring by Satellite. Technical & Operational Support (2003), http://www.esa.int/esapub/bulletin/bullet115/chapter7{_}bul115.pdf
11. Kumar, S., Toshniwal, D.: A data mining approach to characterize road accident locations. Journal of Modern Transportation 24(1), 62–72 (mar 2016), http://link.springer.com/10.1007/s40534-016-0095-5

THREAT INTELLIGENCE

Usando informação sobre IP maliciosos para melhorar a eficácia de um sistema SIEM

João Alves[1], Ivo Rosa[2], Ana Respício[3], and Pedro Rodrigues[2]

[1] LASIGE, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal
joaop.talves@outlook.com
[2] EDP - Energias de Portugal, S.A.
Ivo.Rosa@edp.pt
PedroDias.Rodrigues@edp.pt
[3] Departamento de Informática, e CMAFCIO - Centro de Matemática, Aplicações Fundamentais e Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal
alrespicio@fc.ul.pt

Abstract. Threat Intelligence é um processo de ciberdefesa completo, devido ao uso combinado de mecanismos de descoberta de informação externa e interna. No entanto, o uso de informação externa a partir de fontes públicas, tende a gerar número significativo de alarmes considerados como falsos positivos. A solução apresentada tem como objectivo reduzir esses alertas, continuando a utilizar as fontes públicas. Para atingir esse fim, é utilizado o conceito OSINT para a descoberta e recolha de endereços IP considerados maliciosos por listas públicas. Propõe-se a utilização de métricas de segurança para a avaliação da reputação de endereços IP suspeitos e listas negras, tendo em conta a persistência dos IPs e os incidentes ocorridos na organização. O valor da reputação dos endereços IP é considerado para definir regras a criar num sistema SIEM. A experiência preliminar num ambiente real mostra que a abordagem proposta permite melhorar a eficácia da alarmística de segurança do SIEM.

Keywords: threat intelligence, security metrics, blacklists, open-source intelligence, security information and event management

1 INTRODUÇÃO

Nos últimos anos, face ao aumento em quantidade e em complexidade de ataques informáticos contra diversas organizações, tem-se verificado um crescimento elevado no investimento em plataformas de segurança informática nas infra-estruturas das organizações. As equipas com a responsabilidade de garantir a cibersegurança das organizações necessitam de monitorizar um vasto número de dispositivos, utilizadores, aplicações e, consequentemente, eventos de segurança informática relacionados com esses elementos. A plataforma que é tipicamente utilizada para monitorizar esses eventos de segurança informática é o sistema Security Information and Event Management (SIEM). Este sistema agrega toda a informação de segurança proveniente de diversas fontes, normaliza-a, enriquece-a e envia-a para uma consola centralizada de gestão. A eficiência das

equipas de resposta a incidentes de segurança depende em grande medida da capacidade do SIEM produzir alarmística detalhada e contextualiza sobre possíveis ameaças. Para melhorar essa capacidade é necessário conjugar indicadores externos relevantes com a informação recolhida na infra-estrutura da organização.

Threat Intelligence (TI) é o processo de compreensão das ameaças actuais para as organizações. É necessário estar atento às fontes de informação de cibersegurança para obter indicadores fidedignos e, combinando-os com o conhecimento da infra-estrutura interna, identificar vulnerabilidades existentes antes que estas possam ser exploradas por atacantes. Somente com a conjugação de ambas as fontes de informação é possível ter uma abordagem de TI abrangente e aplicar as medidas de segurança correctas para evitar os ataques aos quais a organização pode estar vulnerável.

A integração de TI em tempo-real no SIEM pode ser materializada em fontes de segurança (*feeds*), das quais se destacam as listas negras (*blacklists*) com informação sobre endereços Internet Protocol (IP), domínios ou URLs suspeitos de actividades maliciosas. A utilização directa das *blacklists*, sem qualquer tipo de pré-processamento, resulta num elevado número de falsos positivos, pela ausência de contextualização e alinhamento com a realidade da organização.

O objectivo do trabalho é aperfeiçoar a alarmística do sistema SIEM, reduzindo a taxa de falsos positivos, com base no nível de confiança nas fontes de informação, e dessa forma contribuir para a eficiência das equipas de cibersegurança nas organizações que usam SIEM. O nosso trabalho consiste no processamento de Threat Intelligence com a conjugação de: SIEM, *blacklists* e métricas de segurança. Com um conjunto de módulos autónomos será possível cumprir três sub-objectivos: (1) Recolher informação sobre endereços IP suspeitos, proveniente de várias *blacklists* públicas; (2) caracterização desses resultados com base em métricas de segurança; (3) integração da informação processada, com um valor de reputação associado, no SIEM, a partir de regras que alertam para comunicações suspeitas na organização, detectando máquinas que estão comprometidas na organização.

Este trabalho está inserido no projecto europeu DiSIEM [4] e resulta da colaboração de dois dos parceiros do projecto, Faculdade de Ciências da Universidade de Lisboa e EDP - Energias De Portugal.

Na secção seguinte analisa-se o trabalho relacionado. Na secção 3 apresentam-se a *framework* proposta e cada um dos seus módulos. O artigo conclui com a discussão dos resultados já obtidos e uma perspectiva de trabalho futuro.

2 TRABALHO RELACIONADO

A utilização de Threat Intelligence numa organização é indispensável, principalmente na actualidade, devido à capacidade de extracção de conhecimento que a equipa de cibersegurança pode obter, e consequentemente, uma melhor eficácia e rapidez na resposta a incidentes de segurança. O trabalho [6] define TI como um processo fundamental para a ciberdefesa de uma organização, sendo composto por dois principais factores de descoberta de informação, externa e interna. A externa caracteriza-se na descoberta de ameaças fora da organização, a partir de *feeds* (redes sociais, blogs, fóruns, comunidades de segurança ou subscrições pagas), partilha de informação pelo governo,

forças policiais, organizações de segurança ou organizações do mesmo sector ou geograficamente próximas, e comunidades de segurança online. Na descoberta interna, o objectivo é obter um conhecimento detalhado sobre o nível de segurança em que a organização se encontra. A detecção de vulnerabilidades do sistema, monitorização e detecção de anomalias de segurança, e desvios do comportamento normal são factores que ajudam na descoberta interna.

Um conceito utilizado para a descoberta externa é o *open-source intelligence* (mais conhecido como OSINT). Em [16] descreve-se a importância do OSINT e capacidade das tecnologias que o usam. Essas tecnologias permitem capturar e fundir de forma inteligente informação da maior fonte de conhecimento (Internet), produzindo um resultado de valor para a organização. Como referido em [6], as *feeds* são uma boa forma de obtenção de informação sobre as ameaças externas e com a utilização do OSINT é possível recolher de diversas *feeds* informação pertinente. Um exemplo dessas *feeds* que contêm informação sobre ameaças e comportamentos maliciosos são as *blacklists*.

O trabalho mais relevante sobre a recolha de múltiplas *blacklists* e possibilidade de configuração especializada por parte das organizações é o IntelMQ [5]. O IntelMQ tem como principal funcionalidade a recolha e processamento de *feeds* de cibersegurança (como por exemplo *logs*, *tweets*, ou, para o nosso caso, *blacklists*) de forma autónoma. Esta ferramenta objectiva possibilitar à equipa de resposta a incidentes de segurança recolher, de uma forma mais eficiente, informação de um conjunto de *feeds*. Contudo, é necessário uma configuração para cada *blacklist* e, caso a informação que se queira recolher seja diferente dos programas padrão, é necessário criar módulos, ou usar módulos semelhantes, para a correcta recolha de informação da fonte pretendida. No entanto e como [6] refere, é necessário apreciar até que ponto as *feeds* são fidedignas e se realmente é possível confiar nelas, para com base na informação obtida implementar mecanismos de ciber defesa.

Existem alguns trabalhos que averiguam a eficácia de *blacklists*, e quais as *blacklists* que num período de tempo fornecem a informação mais fidedigna. As *blacklists* contêm um número significativo de falsos positivos, como é descrito em [11], [14] e [15]. No entanto, sabe-se que o recurso a informação das *blacklists* é uma medida muito usada para a monitorização e detecção de comportamentos maliciosos, [11], [15]. Em [15] foi analisada a eficácia de quatro *blacklists* (NJABL, SORBS, SpamCop e SpamHaus), que reportam endereços IP suspeitos de envios de mail não solicitado. Para a averiguação de falsos e verdadeiros positivos foi utilizado um programa de detecção de mail não solicitado da Oracle [12]. Após a análise de tráfego de emails num ambiente académico (mais de 7,000 computadores) num período de 10 dias, os resultados confirmaram que as *blacklists* contêm um elevado número de falsos positivos.

O trabalho realizado em [11] tem como objectivo compreender o conteúdo de uma *blacklist* e de que forma é recolhida a sua informação. Os autores apresentam dois mecanismos: a detecção de domínios estacionados (conhecidos como *parked domains*) e a detecção de *sinkholes*. Os autores propõem um mecanismo para distinguir os domínios estacionados dos domínios benignos, reduzindo assim um grande número de domínios não benignos presentes numa *blacklist*. Este trabalho também propõe um método para a detecção de *sinkholes*, utilizando uma técnica desenvolvida por eles (*graph-based*), e a sua remoção nas *blacklists*. *Sinkholes* são, por exemplo, servidores que alojavam

domínios maliciosos, mas foram controlados e mitigados por organizações de cibersegurança, que os utilizam para monitorizar a rede e comunicações com este tipo de maliciosidade. Os autores concluem que as *blacklists* apenas contêm cerca de 20% de domínios maliciosos, originando um grande número de falsos positivos. Em ambos trabalhos é complicado afirmar correctamente e ao longo do tempo se a eficácia da informação contida numa *blacklist* irá aumentar ou diminuir.

As métricas de segurança são a ferramenta mais apta para realizar essa análise de uma forma mais precisa e objectiva. Em [7], [8], [9], [13] e [17] salienta-se a importância das métricas de segurança para o conhecimento sobre o estado de segurança de informação de uma organização. Os autores descrevem um conjunto de requisitos e metodologias como guião para a criação, selecção, manutenção e visualização de métricas.

Para esse efeito é necessário Uma ferramenta para extrair, normalizar, filtrar e agregar os eventos de cibersegurança relevantes de uma organização para um sistema de informação de gestão centralizada - SIEM.

Actualmente há vários artigos que providenciam um conjunto de métricas que podem ser utilizadas num sistema SIEM. [2] contem um conjunto de métricas transversais de segurança informação que podem ser aplicadas num sistema SIEM, nomeadamente, o ArcSight [3]. No mesmo artigo apresenta-se a métrica *Quiet Feeds*, que descreve o correcto funcionamento das fontes e que contabiliza o número de fontes que não estão a enviar dados. Esta métrica pode ser usada para a descoberta de informação interna para saber que fontes (*logs*, antivírus, IPS) ligadas aos conectores não estão a enviar informação. Pode também ser usada para identificar quais as *blacklists* que possam não estar a providenciar informação relevante para a organização.

Um trabalho que utiliza o conhecimento exterior (sem considerar OSINT) e conhecimento da organização é o trabalho [10], que cria modelos sobre o impacto de uma ameaça numa organização. Com a utilização de métricas de segurança para identificação de vulnerabilidades nos activos e dependências entre eles, e considerando uma ameaça específica, os autores projectam um modelo que identifica quais os activos da empresa estão sujeitos à ameaça, indicando no fim um valor de risco para cada ameaça. No entanto, isto não é realizado de forma autónoma, sendo necessário uma investigação humana sobre novas ameaças e a recolha de informação sobre as vulnerabilidades da organização.

O OTX da Alienvault [1] é um mecanismo semelhante ao desenvolvido no nosso trabalho. Esta *framework* reúne informação sobre endereços IP, através de denúncias por um conjunto de comunidades. Após a recolha, é feita uma avaliação da ameaça dos endereços denunciados, tendo em conta o número de ataques, o número de denúncias e o tipo de maliciosidade a que o endereço IP suspeito está associado. O resultado é uma lista de endereços IP que pode ser utilizada para a monitorização ou bloqueio dos endereços de IP com um valor de ameaça calculado pelo OTX. No entanto, isto apenas é feito para os endereços IP que estão no OTX, e não por várias *blacklists* escolhidas pela equipa de segurança da organização. Por outro lado, não tem em conta os casos da organização para reavaliar o valor de reputação de cada IP.

Como se pode apreciar pela revisão da literatura relacionada, tanto quanto sabemos, não existem muitos trabalhos, para além do nosso, que façam a ligação usando TI,

entre *blacklists*, métricas de segurança e sistemas SIEM. Em particular não encontrámos nenhum trabalho que se foque na avaliação da qualidade de informação nas *blacklists* de forma contínua e tendo em conta o estado de segurança da organização, assim como o resultado de ocorrência de incidentes face aos alertas provenientes das *blacklists*.

3 DESCRIÇÃO DA FRAMEWORK

O objectivo principal deste trabalho é a redução de falsos positivos em eventos relacionados com comunicações com endereços IP suspeitos de actividade maliciosa. Para tal, procede-se à avaliação da qualidade das *blacklists*, com base no cálculo de métricas de segurança e tendo em conta os casos da organização, relativamente a comunicações entre a organização e os endereços IP suspeitos de actividade maliciosa. As *blacklists* são listas que contêm informações sobre elementos não confiáveis e são a ferramenta típicamente usada como mecanismo de monitorização usado pelas equipas de ciberdefesa [11]. Uma lista de assinaturas de *malware*, usadas pelo antivírus ou sistemas de prevenção de intrusão (IPS), é um exemplo de *blacklists* de assinaturas. Para o nosso caso de estudo, vamo-nos concentrar em *blacklists* que contenham informações relativas a endereços IP suspeitos. Para se obter uma lista de endereços IP mais fidedigna e com uma redução de falsos positivos é necessário classificar a reputação de cada endereço IP e de cada *blacklist*. O cálculo tem de ser contínuo (ou quando há uma alteração) e tem que ter em conta os casos da organização.

A Fig. 1 apresenta uma visão geral da *framework* desenvolvida, que inclui quatro módulos que podem ser usados de forma independente. O primeiro, o Coletor de IP, é um programa OSINT de fontes públicas (*blacklists*). O segundo, o Avaliador da reputação faz uma avaliação da reputação dos endereços IP maliciosos e das *blacklists* que os contêm. O terceiro módulo, a aplicação TABI (Trust Assessment of Blacklists Interface), consiste numa interface web de gestão sobre os endereços IP, *blacklists* e casos relacionados com comunicações entre a organização e endereços IP suspeitos. Finalmente, faz-se a introdução de uma lista reputacional de IP no SIEM e, definem-se as regras para monitorização e geração de alarmes.

3.1 Colector de IP

Para se obter um bom processo de ciberdefesa não se pode apenas considerar o conhecimento que se tem da organização. É estritamente essencial um conhecimento sobre as maliciosidades actuais e que podem comprometer a segurança da organização. Consideremos como fonte uma página de Internet (*feed*) que pode conter uma ou várias *blacklists*, listas que por sua vez identificam, podendo ou não conter informação adicional, *hosts* suspeitos ou conteúdos maliciosos. Neste caso, apenas se pretende considerar *blacklists* que contenham informação sobre endereços IP. A nossa *framework* utiliza o conceito OSINT para reunir informação de um conjunto pré-definido de *blacklists*. A selecção destas realizou-se ao longo de dois meses tendo-se obtido 28 fontes e 121 *blacklists*.

O Colector de IP funciona de forma contínua. O pseudocódigo 1 descreve o processo de recolha e tratamento de informação das *blacklists* pelo Colector de IP. A

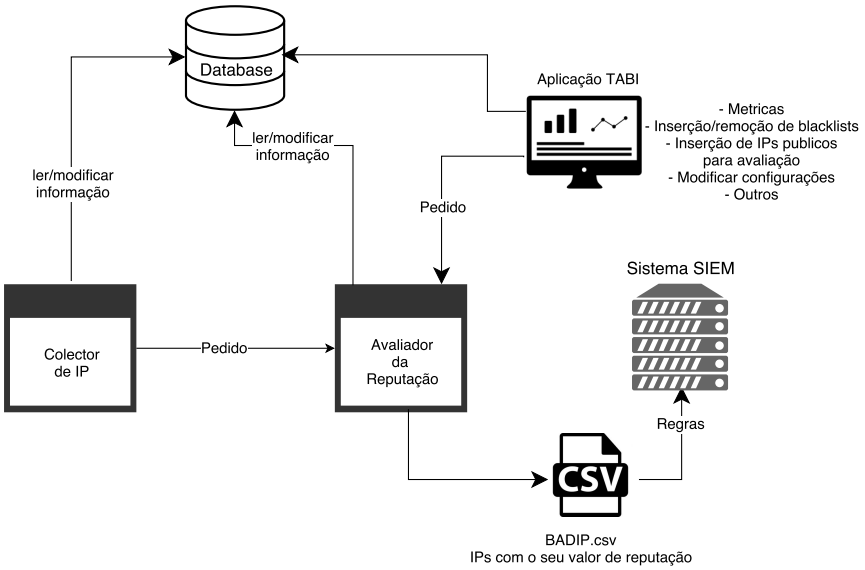


Fig. 1. Modelo do fluxo da framework

informação é recolhida diariamente ao longo do mês, sendo que em cada dia adicionam-se os endereços IP que não tinham sido recolhidos no dia anterior. Para cada *blacklist* são procurados endereços IP válidos. Caso existam são normalizados e verifica-se se já se encontram na lista em construção. Para cada IP que se encontra na lista mas não tenha sido reportado pela *blacklist* em análise, incrementa-se o número de ocorrências desse IP e cria-se uma ligação entre o IP e esta *blacklist*. Caso o IP ainda não se encontre na lista, insere-se na lista, com valor 1 no número de ocorrências e com uma ligação à *blacklist* actual.

Após concluído o processo de recolha por todas as *blacklists* faz-se um pedido de avaliação ao módulo Avaliador da reputação. Depois guarda-se na base de dados a lista de endereços IP e *blacklists* com as respectivas informações. No primeiro dia de cada mês e antes da recolha de informação, actualiza-se o mês e o histórico com a informação do mês que terminou. Faz-se também a actualização da lista de endereços IP, removendo-se aqueles que durante os últimos três meses não tenham sido reportados nas *blacklists* mas que tenham casos positivos no mês que terminou.

3.2 Avaliador da reputação

Para o cálculo da reputação de um endereço IP são utilizadas quatro componentes: $termo\text{-}frequencia$, $precisao$, $media$ da reputação das blacklists que reportaram esse IP e $persistencia$. Os valores da reputação e das restantes métricas variam em [0;100].

A componente $termo\text{-}frequencia$ (TF) é a frequência relativa do IP face ao máximo de ocorrências de algum IP e é calculado dividindo o número de ocorrências do IP pelo número máximo de ocorrências, tendo assim uma relação da relevância do IP.

Algorithm 1 IP collector

```
1: function COLLECTOR()
2:     month
3:     while true do
4:         blacklistsList ← all blacklists in database
5:         ips ← all IP in database
6:         if month not equal to current month then
7:             historic_update(ips,blacklistsList)
8:             month = current month
9:         for each blacklist ∈ blacklistsList do
10:            raw_data = GETDATA(blacklist)
11:            for each line ∈ raw_data do
12:                line_ips ← every IP present in the line
13:                for each ip ∈ line_ips do
14:                    if isIP(ip) then
15:                        normalized_ip = NORMALIZE(ip)
16:                        if normalized_ip ∉ ips then
17:                            ADDIP(ip, ips, blacklist)
18:                        else
19:                            UPDATEIP(ip, ips, blacklist)
20:        ASSESSMENT()          ▷ trustworthiness assessment for the IP and blacklists
21:        INSERTDB()
22:        seconds ← number of seconds for the next day
23:        sleep seconds
```

A $precisao$ de um IP é a razão entre o número de casos confirmados de detecção de *malware* originado por comunicações com esse IP no mês actual e o número total de casos associados a comunicações com esse IP, *ie*, casos positivos e falsos positivos.

A $persistencia$ é definida para um período de três meses e é uma medida da permanência do IP ao longo desse período nas *blacklists* ou em casos positivos. Como um dos objetivos do nosso trabalho é adaptar o programa ao ambiente da organização, quando um IP não tenha sido informado pelas *blacklists* só é descartado se não estiver associado a casos positivos (a precisão é nula). Consideramos o mês atual como sendo o mês 0. Assim, os três meses anteriores serão indexados negativamente com -1, -2 e -3, respetivamente.

O valor da $persistencia$ é obtido pela soma ponderada da permanência do IP ao longo dos meses:

$$Persistencia = Mes_{-1} \times Peso_{-1} + Mes_{-2} \times Peso_{-2} + Mes_{-3} \times Peso_{-3} \times 100 \quad (1)$$

Quando no início do mês se verifica que o IP foi reportado por uma *blacklist* ou que a sua precisão é positiva, o mês terá valor um. Caso contrário o valor será de zero. O peso de cada mês é multiplicado pela presença do IP (0 ou 1). Os pesos foram adaptados por experiência computacional de forma a que $persistencia < 100$.

Durante a análise dos casos da organização verificou-se que existiam endereços IP, que apesar de não estarem a ser reportados pelas *blacklists*, continuavam a surgir em casos da organização, no qual foi confirmado infecção no ativo. Tendo este conhecimento,

achou-se necessário incluir a persistência como um componente de avaliação. Foi usado três meses de persistência porque no inicio deste trabalho os dados disponíveis eram de três meses.

Para o cálculo da reputação de uma *blacklist* são utilizadas duas componentes: $historico$ e $precisao$, cujos os valores variam em [0;100].

A $precisao$ de uma *blacklist* é a razão entre o número de casos confirmados de detecção de *malware* originado por comunicações com os IP reportados pela mesma no mês actual e o número total de casos a que está associada.

A componente $historico$ tem como objectivo avaliar a reputação de uma *blacklist* ao longo do tempo. Esta componente é definida pelo somatório do valor de reputação da *blacklist* do último trimestre, sendo que cada mês tem associado um peso.

$$Historico = Mes_{-1} \times Peso_{-1} + Mes_{-2} \times Peso_{-2} + Mes_{-3} \times Peso_{-3} \times 100 \quad (2)$$

onde Mes_i é a reputação da *blacklist* no mês i.

O cáculo da reputação de uma blacklist é a média das duas componentes.

3.3 Aplicação TABI

A aplicação TABI - Trust Assessment Blacklist Interface - é uma interface web de consola de administração e visualização de informação. A TABI permite ter uma gestão centralizada de toda a *framework*, sem a necessidade de escrita de código para a gestão. A aplicação permite a adição, remoção e edição de *blacklists* e casos de incidente, para serem utilizados na avaliação da reputação dos endereços IP e *blacklists*. A aplicação TABI tem uma funcionalidade extra que é a indicação se um IP público da organização está contido em alguma *blacklist*. Para esta funcionalidade estar operacional é necessário a introdução dos endereços IP públicos da organização.

A TABI também exibe métricas gerais sobre o estado de toda a *framework*, como por exemplo número de endereços IP recolhidos, o número de Quiet Blacklists, os últimos 10 casos ou o top 10 por reputação ascendente de *blacklist* e endereços IP. A Fig. 2 exibe parte da página inical da aplicação com métricas que exibem o funcionamento do sistema. A primeira métrica indica o número que endereços IP que a *framework* recolheu, o número de *blacklists* que não andam a conter informação relativa a endereços IP, o número de casos abertos que estão relacionados com comunicações suspeitas e, o número de endereços IP públicos da organização que estão a ser considerados maliciosos pelas *blacklists* (na imagem aparece a palavra *Clean*, porque nenhum endereço IP público da organização está a ser considerado como malicioso).

Também são apresentadas métricas mais específicas. Para o caso de uma *blacklist*, por exemplo, é apresentada a sua precisão ao longo dos meses, o número de incidentes a que está associada, o número de endereços IP que reportou, o valor da sua reputação e todos os valores usados para o cálculo da reputação. Para um endereço IP algumas métricas são semelhantes, tais como o número de casos de incidente, o número de casos falsos e verdadeiros positivos, o número de *blacklists* que o reportaram, o seu valor de reputação, a sua classificação em comparação com os outros endereços IP e todos os valores usados para o cálculo da reputação. A informação sobre casos inclui métricas

Fig. 2. Conjunto de métricas que indicam de uma forma geral o estado da *framework*

relativas aos endereços IP e blacklists associadas, se o caso é positivo ou falso positivo, entre outros. A Fig. 3 apresenta um gráfico de variação da precisão de uma determinada *blacklist* ao longo do tempo, conforme se obtem na aplicação.

Os valores considerados são valores fictícios e não representam nenhum caso real.

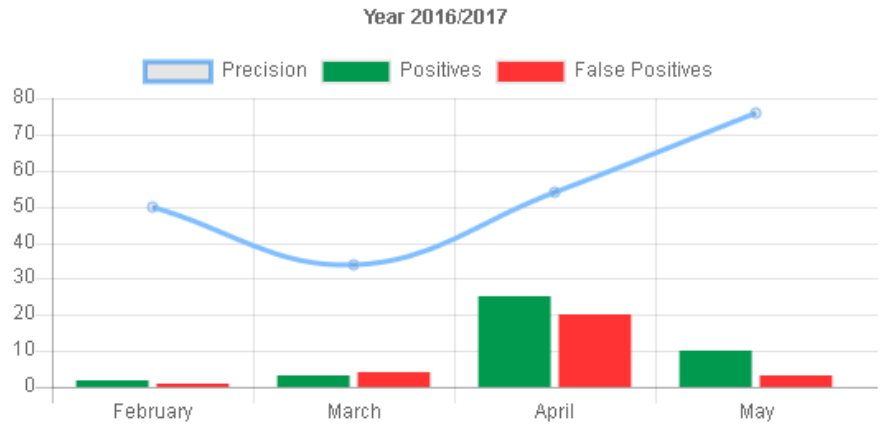


Fig. 3. Precisão de uma *blacklist*

3.4 Regras

Uma regra do SIEM é um procedimento programado que avalia eventos de entrada para condições e padrões específicos. Quando ocorre um evento para o qual existe uma correspondência com uma regra, desencadeiam-se acções em resposta, tal como um alarme. Isto ajuda a equipa de segurança de resposta a incidentes a analisar e monitorar tipos específicos de eventos.

Para que haja monitorização e alerta de comunicações com endereços IP suspeitos de maliciosidade e para detectar ativos da organização que se encontram infectados e estão a tentar comunicar com redes de botnet, é necessário criar regras no SIEM que alertam quando um endereço da organização realiza uma, ou mais, comunicações com um, ou mais, endereços IP suspeitos de actividade maliciosa.

No caso de aplicação, as regras não irão apenas servir para monitorizar e alertar, mas também para a redução de falsos positivos. Sempre que é realizada uma modificação no valor de reputação de um endereço IP, o ficheiro que contem os endereços IP e o valor de reputação associado é alterado (ficheiro BADIP na Fig. 1). O conjunto de regras do SIEM irá recolher cada conjunto de endereços IP que contenham o intervalo de valor de reputação que a regra está destinada a tratar e irá monitorizar esses endereços IP na rede da organização.

O alarme de uma regra depende da afinação de vários parâmetros: o intervalo de tempo de observação, o número de comunicações nesse intervalo de tempo, e o número de diferentes endereços IP suspeitos nessas comunicações. A definição destes parâmetros deve considerar o nível de reputação de cada IP de forma a reduzir o número de falsos positivos e ter uma maior confiança nos alarmes disparados por estas regras.

4 CONCLUSÕES E TRABALHO FUTURO

A *framework* desenvolvida tem sido refinada e tem estado em avaliação ao longo de quatro meses. Atualmente consideram-se 121 *blacklists* para a recolha e avaliação, em média, de 166750 endereços IP. A experiência de avaliação consistiu em (1) manter a lista que estava em utilização pela organização para detecção de comunicações com endereços IP maliciosos (lista original); (2) recolher os casos resultantes destas comunicações e de outros ataques de código malicioso; e (3) comparar os casos de incidente com o resultado que decorreria da utilização da lista produzida pela nossa abordagem (lista nova).

As conclusões que se apresentam de seguida são preliminares e resultam dos quatro meses de avaliação. Relativamente à precisão de detecção de comunicação com endereços IP maliciosos, observou-se que a nossa solução apresenta, em média, 60,66%, uma melhoria de 3% em comparação com a utilização de listas públicas sem tratamento e sem considerar os casos da organização. Este aumento deve-se à utilização de componentes em consideração com os incidentes tratados pelas equipas de resposta a incidentes de segurança, como o caso da componente $persistencia$.

No início do último mês da experiência aqui relatada, começou-se a testar a criação de regras com alertas adaptativos, ajustando o limiar dos alertas de acordo com a pontuação de reputação do endereço IP. Acreditamos que isso terá um impacto positivo na redução da taxa de falsos positivos.

Como se pode verificar pelas conclusões, a avaliação da reputação dos endereços IP e *blacklists* já demonstrou resultados positivos e que justificam a sua utilização. No entanto, o trabalho não está ainda numa fase final e é ainda possível melhorar a avaliação da reputação e reduzir o volume de informação reportado pelas *blacklists*. Para a redução de volume o que se pode utilizar é a detecção de *parked domains*, como descrito em [11]. De momento não se está a pensar utilizar a detecção de *sinkholes*, também descrito no artigo como outro método de redução de volume de informação, porque existe a possibilidade de haver máquinas a comunicar com *sinkholes* e estarem infectadas. Isto acontece porque apesar de o *sinkhole* não infectar a máquina que está em comunicação com ele, nada impede que a máquina já tenha sido infectada e está a tentar comunicar com um conjunto de endereços IP em que o *sinkhole* está inserido.

Como o nosso objectivo é a detecção de casos de infecção, ao excluir esta comunicação estar-se-ia a excluir casos positivos. Para aperfeiçoar a avaliação da reputação, perspectivamos a introdução de pesos para cada componente no modelo de avaliação da reputação e a introdução de um parâmetro sobre o tipo de maliciosidade (por exemplo, *ransomware*, *phishing*, ou C&C) a que um endereço IP possa estar associado. Esta classificação, sobre o tipo de maliciosidade, poderá ser um critério a ser aplicado para definição de novas regras nos sistemas SIEM.

5 Agradecimentos

Este trabalho foi parcialmente financiado pelo projecto DiSIEM, financiado pela Comissão Europeia no programa H2020, G.A. nº 700692, e pela FCT através do projecto UID/MAT/04561/2013. Os autores agradecem a colaboração de Gonçalo Martins.

References

1. AlienVault. (2016). AlienVault Open Threat Exchange (OTX) TM User Guide, 1–44.
2. ArcSight. (2010). WhitePaper: Security Operations Metrics Definitions for Management and Operations Teams. HP ArcSight, 44(0), 0–7.
3. ArcSight ESM. (2017). Retrieved June 29, 2017, from https://saas.hpe.com/en-us/software/siem-security-information-event-management.
4. DiSIEM. Retrieved June 27, 2017, from http://disiem-project.eu/.
5. Enisa, CNCS, CERT-EU, CERT.AT, and CERT.BE (08 Feb. 2016). "Incident Handling Automation." Incident Handling Automation - Enisa. Retrieved June 27, 2017, from https://www.enisa.europa.eu/topics/csirt-cert-services/community-projects/incident-handling-automation.
6. Bromiley, M. (2016). Threat Intelligence: What It Is, and How to Use It Effectively. SANS Institute.
7. Jansen, W. (2009). Directions in Security Metrics Research, NISTIR 7564. National Institute of Standards and Technology, April, 1–26.
8. Jaquith, A. (2007). Security Metrics: Replacing Fear, Uncertainty, and Doubt. Statewide Agricultural Land Use Baseline 2015 (Vol. 1). Addison-Wesley. https://doi.org/10.1017/CBO9781107415324.004.
9. Julisch, K. (2009). A Unifying Theory of Security Metrics with Applications with Applications. Security, 19.
10. Kotenko, I., Polubelova, O., Saenko, I., & Doynikova, E. (2013). The ontology of metrics for security evaluation and decision support in SIEM systems. Proceedings - 2013 International Conference on Availability, Reliability and Security, ARES 2013, 638–645. https://doi.org/10.1109/ARES.2013.84.
11. Kührer, M., Rossow, C., & Holz, T. (2014). Paint it black: Evaluating the effectiveness of malware blacklists. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).
12. Bailey, M., Oberheide, J., Andersen, J., Mao, Z., Jahanian, F., & Nazario, J. (2007) Automated classification and analysis of internet malware. In Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID'07)
13. Payne, S. C. (2006). A Guide to Security Metrics. SANS Institute, 36.
14. Rossow, C., Czerwinski, T., Dietrich, C. J., & Pohlmann, N. (2010). Detecting Gray in Black and White. MIT Spam Conference.

Decomposição automática de processos de negócio dependentes da IoT com distribuição de dados e fluxo de controlo

Daniel Vitoriano, Dulce Domingos e Francisco Martins

LASIGE, Faculdade de Ciências, Universidade de Lisboa
Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa,
1749-016 Lisboa, Portugal
dvitoriano@lasige.di.fc.ul.pt,{mddomingos, fcmartins}@ciencias.ulisboa.pt

Resumo. Quando utilizada nos processos de negócio, a Internet das Coisas (IoT) é geralmente encarada como uma plataforma distribuída de recolha de informação. No entanto, a IoT tem capacidade computacional que pode e deve ser utilizada de forma a maximizar a sua autonomia energética, reduzir a quantidade de informação trocada e executar partes dos processos. De facto, os processos de negócio ainda são definidos seguindo uma abordagem centralizada, dificultando o uso das capacidades destes dispositivos. Neste artigo apresentamos uma solução automática de decomposição de processos de negócio BPMN 2.0 (*Business Process Model and Notation*) dependentes da IoT. Partimos de um processo de negócio que segue a abordagem centralizada e, aplicando um método de decomposição, transferimos para os dispositivos da IoT as operações que podem ser aí realizadas, tendo em conta o fluxo de controlo e o fluxo de dados. Esta transformação permite reduzir o processamento central e o número de mensagens trocadas da rede. O código que será executado pelos dispositivos da IoT é gerado automaticamente a partir do BPMN.

Palavras-chave: Processos de negócio, IoT, BPMN, decomposição.

1 Introdução

A IoT é uma das dimensões chave da nova revolução industrial, motivada pelo rápido desenvolvimento das tecnologias digitais e por uma mudança de paradigma nas metodologias de negócio e no quotidiano das pessoas. É caracterizada pela presença de objetos, tais como etiquetas RFID (*RadioFrequency IDentification*), sensores, atuadores, dispositivos móveis, entre outros, que, através de esquemas de endereçamento único, são capazes de interagir entre si e de cooperar com os seus vizinhos de modo a atingir um objetivo comum. É um paradigma que torna os objetos que nos rodeiam em atores ativos da Internet, gerando e consumindo informação. A literatura cientifica tipicamente procura colmatar os seguintes problemas provenientes destes objetos: eficiência energética (que é o recurso mais escasso nas redes de sensores), escalabilidade (visto que o número de dispositivos pode ser bastante elevado), fiabilidade (pois

a rede pode ser utilizada para reportar eventos do tipo alarme) e robustez (já que os nós da rede estão suscetíveis ao mais variado tipo de falhas) [16, 17, 18, 19, 20].

Um modelo de processo é um modelo conceptual de como os negócios realizam as suas operações, através da especificação de atividades, eventos, estados, lógica de fluxo de controlo, entre outros fatores. Os modelos de processo são críticos para a otimização e automatização de processes de negócio e são muitas vezes representados numa notação gráfica, como é o caso do BPMN [14]. Estes processos podem utilizar as capacidades que os dispositivos IoT possuem para ler e atuar sobre o meio circundante ao processo. Inclusivamente, podem aproveitar a IoT para executar parte do seu fluxo de execução quando devidamente decompostos.

A decomposição passa por partir um sistema em subsistemas progressivamente mais pequenos que são responsáveis por alguma parte do problema do domínio. A decomposição é também um meio para modelar grandes sistemas e para facilitar a reutilização de modelos parciais, porque reduz o acoplamento, favorece a compactação de especificações e permite que vários níveis de detalhe possam coexistir. Como consequência, os modelos tornam-se mais fáceis de compreender o que, por sua vez, facilita a sua validação, redesenho e otimização [13].

A nossa solução decompõe de forma automática processos de negócio, transformando o processo original noutro que aproveita os recursos computacionais que os dispositivos IoT dispõem. Começamos por gerar um grafo, a partir de um modelo BPMN, que capture as dependências de fluxo de controlo e de dados das tarefas. Depois, identificamos os caminhos que só contêm elementos BPMN capazes de serem executados em dispositivos IoT e que poderão ser transferidos para esta rede. Em seguida, com base nestes caminhos, verificamos quais se enquadram nos padrões que identificámos e aplicamos o procedimento de transformação correspondente. Por último, redesenhamos o modelo BPMN com a solução encontrada. Esta solução permite reduzir o número de comunicações realizadas com os dispositivos IoT.

Este artigo está organizado da seguinte forma: a secção seguinte apresenta o trabalho relacionado; a Secção 3 apresenta a nossa abordagem de decomposição de processo através de um caso de uso que serve como motivação do nosso procedimento; a Secção 4 mostra o protótipo desenvolvido; a última secção termina o artigo com conclusões e trabalho futuro.

2 Trabalho Relacionado

O projeto Mentor apresenta uma das primeiras propostas sobre decomposição de processos [11]. Os processos são definidos através de diagramas de estados e a sua decomposição em partições tem em conta o fluxo de controlo e o fluxo de dados.

A utilização crescente da *Web Services Business Process Execution Language* (WS-BPEL) [8] justificou o desenvolvimento de propostas sobre decomposição de processos WS-BPEL. Nanda el al. [7] criam novos subprocessos para cada serviço do processo original, os quais comunicam diretamente evitando que a sincronização e a troca de mensagens sejam da exclusiva responsabilidade do motor de execução de

processos central. No entanto, estes autores não consideram a hipótese de agrupar serviços no mesmo subprocesso.

Sadiq el al. [10] e Fdhila et al. [5] decompõem modelos de processos genéricos agrupando várias atividades em subprocessos e distribuindo a execução destes subprocessos por diferentes motores de execução de processos. Sadiq et al. apenas consideram as dependências do fluxo de controlo, enquanto que Fdhila et al. também consideram as dependências do fluxo de dados. Em [4], Fdhila et al. otimizam a composição dos subprocessos tendo em conta vários parâmetros de qualidade de serviços, tais como, custo, tempo, fiabilidade e disponibilidade. Yu et al. [12] utilizam programação genética para criar partições (subprocessos) de processos WS-BPEL que utilizam dados de forma intensiva.

Com o objetivo de usufruir das vantagens oferecidas pela nuvem para a execução de partes dos processos de negócio, Duipmans et al. [3] dividem os processos de negócio em duas partes: a que é executada localmente e a que é executada na nuvem. Com esta divisão, os autores pretendem executar na nuvem as tarefas computacionalmente mais intensas e cujos dados não tenham requisitos de confidencialidade. A identificação destas tarefas é feita manualmente. Povoa et al. [9] propõem um mecanismo semiautomático para determinar a localização das atividades e dos respetivos dados baseado em políticas de confidencialidade, custos monetários e métricas de desempenho. Hoenisch et al. [6] otimizam a distribuição das atividades tendo em conta alguns parâmetros adicionais tais como o custo associado a atrasos na execução de atividades e o tempo não usado, mas pago, de recursos da nuvem. Yousf et al. tiram partido da computação ubíqua e definem processos de negócio ubíquos, como forma de melhorar o desempenho prestado pelos processos de negócio e estendem o BPMN para que este suporte requisitos adicionais que são específicos da computação ubíqua, de forma a permitir a modelação de processos de negócio ubíquos [1].

Em [2], apresentamos uma proposta preliminar ao problema de decomposição de processos de negócio BPMN de modo a que partes desses processos possam ser executados em dispositivos IoT. A decomposição é baseada em tabelas de dependências tendo em conta apenas o fluxo de controlo. A identificação das partes do processo de negócio que podem ser incluídas em partições a serem executadas pelos dispositivos IoT é feita automaticamente, tendo em conta as capacidades destes dispositivos. O trabalho apresentado neste artigo considera adicionalmente as restrições derivadas do fluxo de dados. As partições a serem executadas nos dispositivos IoT podem incluir também dados não persistentes do processo. O procedimento de decomposição dos processos prescinde das tabelas de dependências, reduzindo a memória e a computação utilizadas. Os caminhos candidatos a partições são gerados diretamente do grafo criado a partir do modelo definido em BPMN.

3 Decomposição de Processos de Negócio

Esta secção descreve a nossa solução de decomposição de processos de negócio com foco na redução de comunicações entre dispositivos IoT e o sistema central. Para

ilustrar, utilizamos um processo de negócio de um sistema de rega automática simplificado que serve como caso de uso.

3.1 Caso de uso

Uma organização utiliza um sistema de rega automática que controla o nível de água presente nos tanques e verifica a necessidade de iniciar uma rega pela humidade do solo. Neste sistema, sempre que é lido o valor do nível de água dos tanques na rede de dispositivos IoT, é feito um pedido para repor a água em falta. Desta forma, existe a garantia que as irrigações decorrerão com a quantidade máxima de água disponível.

O valor de água reposta é guardado num histórico para futura auditoria de despesas. Além disso, o sistema contacta periodicamente a rede de dispositivos IoT para saber a humidade do solo. Caso esta esteja abaixo dos valores pré-estabelecidos, é enviado um sinal para a rede de dispositivos IoT que desencadeia o processo de rega. Os valores de humidade do solo são mantidos num histórico, de forma a ajustar o intervalo de tempo entre comunicações com a rede de dispositivos IoT consoante o período climático.

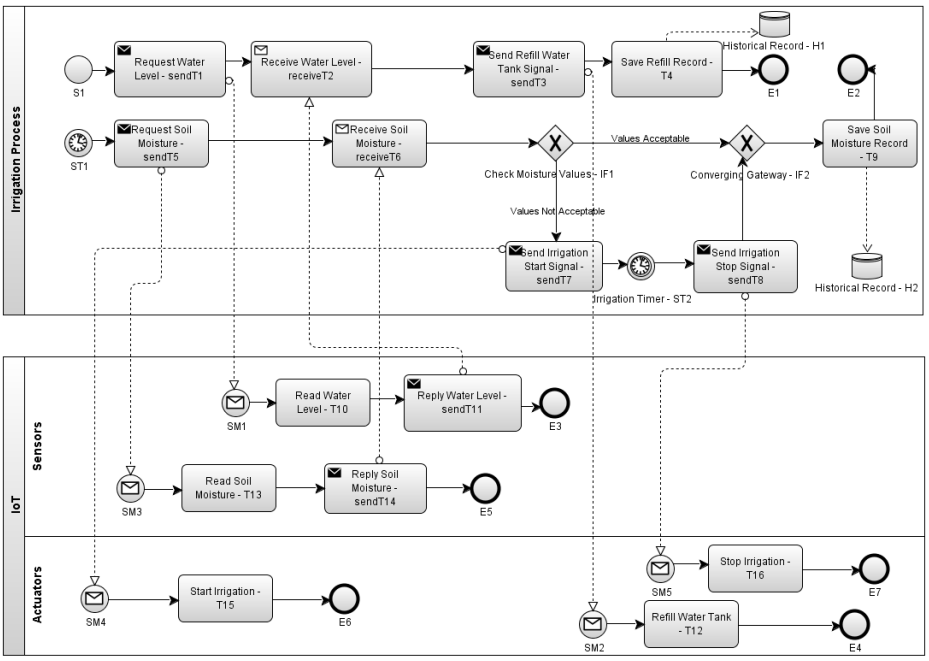


Fig. 1. Modelo BPMN de um sistema de rega automática

A Figura 1 ilustra o modelo BPMN simplificado deste caso de uso. É composto por dois participantes: o processo de irrigação em cima (*Irrigation Process*) e a rede de dispositivos IoT. O processo de irrigação representa o servidor central responsável

pela execução dos processos de negócio. Contém dois fluxos de execução: o de cima descreve o reabastecimento de água num tanque e o outro define o processo de rega. A rede de dispositivos IoT possui um conjunto de sensores e atuadores.

O primeiro fluxo de execução começa com o evento de início *S1*. Este desencadeia a tarefa de envio de mensagem *Request Water Level* que contacta os sensores para obter informação sobre o nível de água. Para isso, envia uma mensagem para os sensores (*SM1*) que dá início à execução do processo localmente. Os sensores leem o nível de água que está no tanque (*Read Water Level*) e encaminham esta informação de volta para o processo de irrigação (*Reply Water Level)*, terminando em seguida o processo (sinalizado pelo evento de fim *E3*). A tarefa de receção de mensagens *Receive Water Level* aguarda a chegada da mensagem dos sensores. Quando esta chega, desencadeia a tarefa *Send Refil Water Tank Signal* que envia uma mensagem (*SM2*) com a informação recebida para os atuadores. Esta informação é transmitida à tarefa *Refill Water Tank* para encher o tanque com a quantidade de água em falta, terminando em seguida o processo (com o evento de fim *E4*). Por último, a tarefa *Save Refill Record* regista esta ocorrência escrevendo na *data store Historical Record – H1,* terminando com o evento de fim *E1*.

O segundo fluxo executa periodicamente o evento de início *ST1*. Este fluxo pretende detetar se os níveis de humidade do solo fornecidos pelos sensores são aceitáveis. Para isso, *Request Soil Moisture* contacta os sensores a solicitar esta informação. O evento de início *SM3* começa o processo e a tarefa *Read Soil Moisture* lê dos sensores a informação sobre a humidade do solo. Em seguida, *Reply Soil Moisture* envia uma mensagem com esta informação para o processo de rega e o processo termina. Depois de a mensagem ter sido entregue à tarefa de receção de mensagens *Receive Soil Moisture*, é feita uma análise à informação recebida de forma a verificar se é necessário iniciar uma rega. Para isso, é utilizado o *gateway* exclusivo *Check Moisture Values* que contém a condição sobre os valores considerados aceitáveis e que obriga ao processo a seguir apenas um dos caminhos. Caso os valores não sejam aceitáveis, *Send Irrigation Start Signal* sinaliza os atuadores para darem início à irrigação, através da tarefa *Start Irrigation*. O processo termina com a execução desta tarefa. É assim necessário o envio de outro sinal para que a irrigação em curso termine. O objetivo do evento intermédio de tempo *Irrigation Timer* é esperar uma duração pré-estabelecida de tempo antes de a tarefa de envio de mensagem *Send Irrigation Stop Signal* sinalizar os atuadores a terminar a irrigação em curso com a tarefa *Stop Irrigation*. Por fim, é feito um registo pela tarefa *Save Soil Moisture Record* na *data store Historical Record - H2* para assinalar se na sequência deste fluxo ocorreu irrigação. O *gateway* exclusivo *Converging Gateway* tem o propósito de remeter o processo para esta tarefa, independentemente do caminho seguido pelo processo.

Este processo segue uma abordagem centralizada e será utilizado nas secções seguintes para exemplificar os vários passos do procedimento de decomposição.

3.2 Procedimento de decomposição

A solução desenvolvida decompõe os processos de negócio transferindo parte do processo para a rede de dispositivos IoT com o objetivo de reduzir o número de comunicações. Para isso, realizamos os seguintes passos:

1. Gerar um grafo, a partir de um modelo de processos de negócio BPMN, que capture as dependências de fluxo de controlo das tarefas.
   A Figura 2 ilustra o grafo que captura o fluxo de controlo do modelo BPMN do caso de uso. A semântica é que uma tarefa está pronta a ser executada após as tarefas antecessoras concluírem. Por exemplo, a tarefa *sendT1* executa após o evento de início *S1*, enquanto que a tarefa *receiveT2* executa após a tarefa sendT1 e após a tarefa *sendT11*. Cada nó corresponde a um par que contém um elemento BPMN e o nome do processo correspondente.

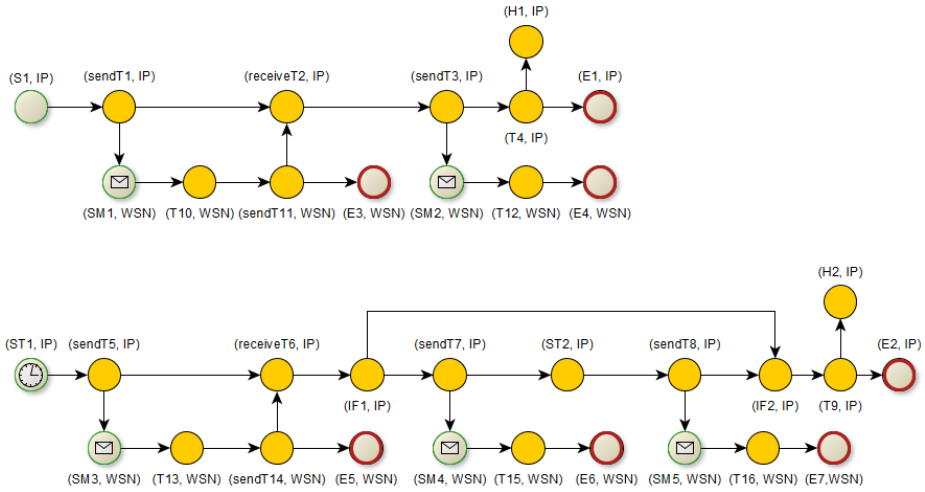


Fig. 2. Grafo resultante do modelo BPMN de rega automática

2. Gerar uma lista de subcaminhos (designados caminhos candidatos) que podem ser transferidos para a rede de dispositivos IoT.
   Para cada nó inicial do grafo, é feita uma travessia enumerando todos os caminhos que contenham arestas de fluxo de mensagens (mensagens trocadas entre dois participantes, isto é, *pools*). Filtram-se os nós que correspondem a elementos BPMN que fazem parte das redes de dispositivos IoT e os elementos BPMN (eventos de tempo e *gateways*) que podem ser executados nos dispositivos. Por fim, removem-se os caminhos cujos nós estejam contidos em caminhos de maior dimensão. Para o nosso caso de uso, obtemos os seguintes caminhos candidatos:

– *sendT1* → *SM1* → *T10* → *sendT11* → *receiveT2* → *sendT3* → *SM2* → *T12* → *E4*.
– *ST1* → *sendT5* → *SM3* → *T13* → *sendT14* → *receiveT6* → *IF1* → *sendT7* → *ST2* → *sendT8* → *SM5* → *T16* → *E7*.

3. Redefinir os participantes com base nos caminhos candidatos obtidos.
   Os participantes do processo são redefinidos ao transferir atividades para a rede de dispositivos IoT ou pela eliminação de comunicações. Identificámos três cenários que requerem redefinição:

– Existe uma tarefa de receção de mensagem seguida de uma tarefa de envio de mensagem e o antecessor e sucessor, respetivos destas tarefas, pertencem ao mesmo participante. Isto representa uma comunicação desnecessária. A transformação passa por eliminar a tarefa de envio de mensagem, o seu destinatário e a comunicação que os une. A Figura 3 mostra a aplicação desta regra. No nosso caso de uso, *receiveT2* antecede *sendT3* e, como *sendT11* e *SM2* fazem parte da rede de dispositivos IoT é possível aplicarmos este caso. *SendT3* e *SM2* são removidos e, de modo a manter o fluxo de controlo, *receiveT2* é ligado a *T4* e *sendT11* é ligado a *T12* (devido a esta ligação *E3* é também removido).

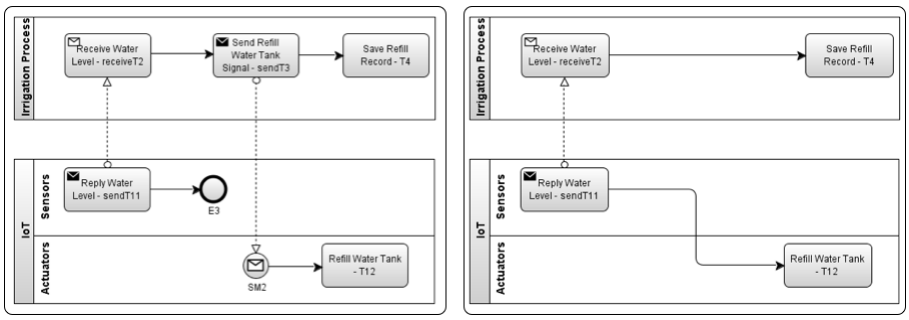


Fig. 3. Caso que demonstra a existência de uma comunicação desnecessária

– O evento de início de um dado processo é um evento de tempo seguido de uma tarefa de envio de mensagem e de uma tarefa que receção de mensagem. Neste caso, a tarefa de envio, a comunicação e o seu destinatário são eliminados e o evento de tempo é movido para a rede de dispositivos IoT. É possível observar na Figura 4 que *ST1* antecede *sendT5* que, por sua vez, antecede *receiveT6*. A transformação a aplicar é eliminar *sendT5* e substituir *SM3* por *ST1*.

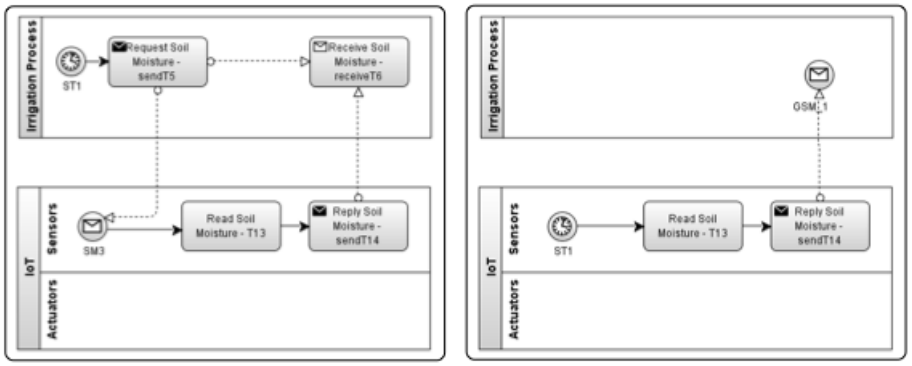


Fig. 4. Caso que ilustra um evento de tempo a transferir para a rede de dispositivos IoT

— Existe uma tarefa de envio de mensagem que antecede uma tarefa de receção ou evento de início de mensagem (de outro participante), que por sua vez antecede um *gateway*. Tipicamente, os sensores têm capacidade computacional suficiente para realizar operações lógicas, pelo que neste cenário é possível transferir os *gateways* para a rede de dispositivos IoT. De modo a manter o fluxo de controlo prévio, as transformações a aplicar quando o *gateway* é transferido dependem dos elementos que o sucedem. A Figura 5 ilustra a aplicação deste cenário no nosso caso de uso. *SendT14* envia uma mensagem a *receiveT6* que antecede o *gateway IF1*. Estas duas tarefas são removidas (assim como *E5*) e *IF1* é transferido para a rede de dispositivos IoT e ligado à tarefa antecessora de *sendT14* (*T13*). Para manter o fluxo de controlo prévio, é necessário decidir o que acontece aos elementos que permaneceram no processo de irrigação e que dependiam de *IF1*. No primeiro ramo, temos *sendT7* que é uma tarefa de envio e, por isso, pode ser movida para a rede de dispositivos IoT. Como tal, procuramos a primeira tarefa que sucede *sendT7* na rede de dispositivos IoT. A operação passa por eliminar *sendT7*, *SM4* e ligar *IF1* a *T15*. *ST2* pode também ser transferido para a rede de dispositivos IoT (por se tratar de um evento intermédio de tempo) e é ligado a *T15*. É aplicado o mesmo procedimento que foi feito a *sendT7* a *sendT8*. Neste caso, a primeira tarefa encontrada na rede de dispositivos IoT a partir de *sendT8* foi *T16* e, portanto, esta é ligada a *ST2*. Contudo, *sendT8* antecedia outro elemento (*IF2*). Para manter o fluxo de controlo prévio, o procedimento utilizado é criar uma tarefa de envio de mensagem na rede de dispositivos IoT (neste caso, *generatedTask_1* que vai suceder *T16*) e o respetivo evento de início de mensagem no participante de destino (neste caso, *GSM_1* que vai anteceder *IF2*). É também criado um evento de fim (*GE_2*) que é ligado à *generatedTask_1* para terminar o processo na rede de dispositivos IoT. Quanto ao segundo ramo, como este começa por um *gateway* (*IF2*), nenhuma ação é tomada, porque o fluxo de controlo nunca vai ser quebrado por possuir elementos antecessores que já foram tratados.

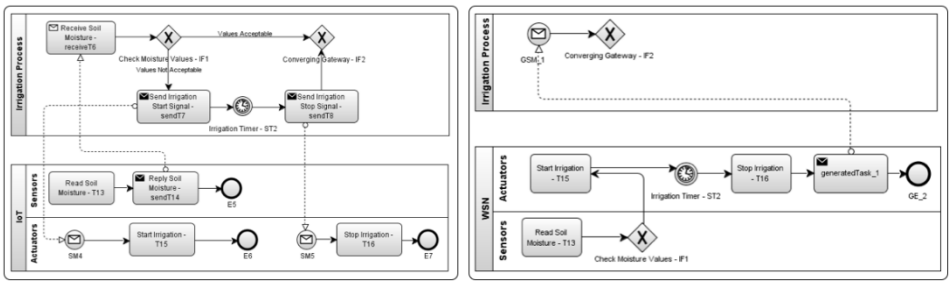


Fig. 5. Caso que ilustra um *gateway* que pode ser movido para a rede de dispositivos IoT

4. Criar o novo modelo BPMN com as alterações efetuadas.
   Utilizamos a aplicação Graphviz [15] para gerar as coordenadas dos elementos do modelo BPMN modificado. A Figura 6 ilustra o resultado final da execução consecutiva dos passos de decomposição até não ser possível ocorrer mais otimizações.

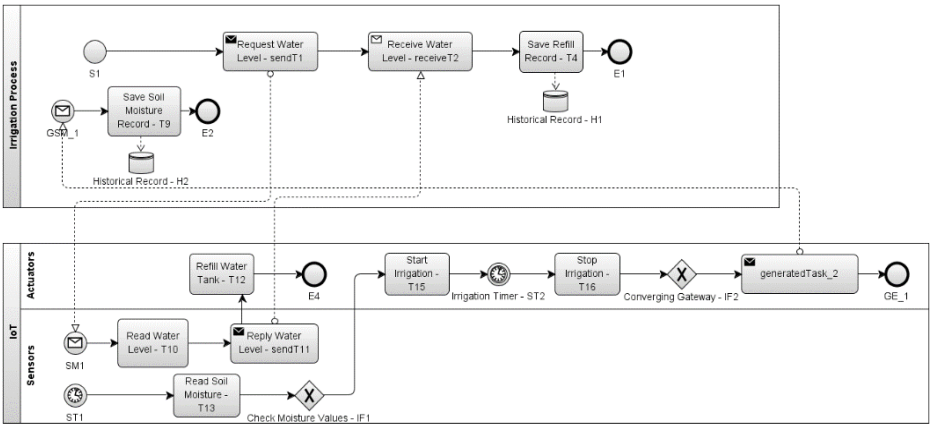


Fig. 6. Modelo BPMN após a decomposição

É possível observar que, das 7 comunicações realizadas entre o processo de irrigação e a rede de dispositivos IoT no modelo original, a solução ótima mostra que apenas 3 são necessárias se tirarmos proveito das capacidades que a rede IoT oferece. O processo de negócio que escolhemos corresponde a uma simplificação de um processo de rega, cujo objetivo é ilustrar as transformações descritas nesta secção. Por esta razão, este processo, após transformado, perde em parte a capacidade explicativa inicial (ver Figura 1).

4 Protótipo

Concebemos um protótipo que realiza os procedimentos de decomposição descritos na secção anterior, cujo código fonte se encontra disponível no github no endereço https://github.com/fcmartins/bpmn-decomposition.git.

O nosso ambiente de desenvolvimento utiliza a linguagem de programação Java e as seguintes ferramentas:

- jBPM (versão 6.3.0);
- Eclipse Luna (versão 4.4.2) com os *plugins* BPMN2 *Modeler* e SonarLint;
- Graphviz.

O jBPM é uma aplicação JavaEE que implementa a norma BPMN. A esta aplicação está associada a versão Luna do Eclipse. O BPMN2 *Modeler* é o *plugin* de visualização gráfica de modelos BPMN e o SonarLint é o *plugin* que utilizamos para garantir a qualidade do código fonte. O Graphviz é uma ferramenta para desenho de grafos especificados na linguagem DOT.

Para executar o protótipo fornecemos como argumento um ficheiro BPMN correspondente ao modelo que se pretende decompor (tipicamente, localizado na diretoria BPMN-Files\examples\ do protótipo). Um dos desafios na implementação, além da decomposição do modelo, foi o de calcular as coordenadas dos elementos BPMN no modelo gerado pelo protótipo. Para isso, recorremos à ferramenta Graphviz. Geramos um ficheiro DOT por cada participante do modelo BPMN e passamos ao Graphviz, que através de um algoritmo de disposição de grafos, gera as coordenadas que iremos utilizar para o desenho da solução. Em seguida, aplicamos algumas transformações geométrica às coordenadas geradas para ficarem de acordo com as restantes componentes do modelo.

Avaliámos o protótipo recorrendo a vários modelos que construímos. Em particular, a decomposição apresentada neste artigo resulta da aplicação do protótipo.

5 Conclusões e Trabalho Futuro

Os processos de negócio usam cada vez mais informação disponibilizada por dispositivos IoT, de modo a darem respostas atempadamente de acordo com o contexto. No entanto, os dispositivos IoT possuem capacidade computacional suficiente para executar partes de processos de negócio, reduzindo assim o processamento central e o número de mensagens trocadas e, consequentemente, aumentando a autonomia energética destes dispositivos.

Tendo em conta que, normalmente, a modelação de processos de negócio continua a seguir uma abordagem centralizada, o trabalho proposto neste artigo permite a decomposição automática de processos de negócio dependentes da IoT. A decomposição tem em conta as restrições impostas pelo fluxo de controlo e pelo fluxo de dados.

Neste momento, este trabalho está a ser integrado com um tradutor de BPMN para código que pode ser executado em dispositivos IoT, com o objetivo de auxiliar as várias fases de desenvolvimento de processos de negócio: definição (com eventual decomposição), instalação (tradução e instalação do código traduzido nos dispositivos) e execução (assegurando a comunicação entre o motor central de execução de processos e os dispositivos). Outro trabalho que nos propomos é o de realizar uma avaliação quantitativa ao procedimento de decomposição que apresentamos. A abordagem também poderá ser generalizada de modo a suportar outras variantes de pro-

cessos de negócio além da rede IoT (por exemplo, modelos ontológicos de processos de negócio).

Referências

1. Yousf, A., Freitas, A., Dey K., A., Saidi, R.: *The Use of Ubiquitous Computing for Business Process Improvement. In: Transactions on Services Computing*, vol. 9(4), pp. 621-632. IEEE (2016).
2. Domingos, D., Martins, F., & Caiola, L.: *Decentralising Internet of Things Aware BPMN Business Processes*. *In: International Conference on Sensor Systems and Software*, pp. 110-119. Springer (2014).
3. Duipmans, E. F., Pires, L. F., & da Silva Santos, L. O. B.: *Towards a BPM cloud architecture with data and activity distribution. In:* IEEE 16th *International Conference on Enterprise Distributed Object Computing Conference Workshops* (EDOCW), pp. 165-171. IEEE (2012).
4. Fdhila, W., Dumas, M., Godart, C., & García-Bañuelos, L.: *Heuristics for composite web service decentralization*. *Software & Systems Modeling*, vol. 13(2), pp. 599-619 (2014).
5. Fdhila, W., Yildiz, U., & Godart, C.: *A flexible approach for automatic process decentralization using dependency tables*. *In*: IEEE *International Conference on Web Services* (ICWS), pp. 847-855. IEEE (2009).
6. Hoenisch, P., Schuller, D., Schulte, S., Hochreiner, C., & Dustdar, S.: *Optimization of complex elastic processes*. IEEE *Transactions on Services Computing*, vol. 9(5), pp. 700-713 (2016).
7. Nanda, M. G., Chandra, S., & Sarkar, V.: *Decentralizing execution of composite web services*. *In*: ACM *Sigplan Notices*, vol. 39(10), pp. 170-187. ACM (2004).
8. OASIS.: *Web services business process execution language version* 2.0. *Technical report, Organization for the Advancement of Structured Information Standards* (2007).
9. Povoa, L. V., de Souza, W. L., Pires, L. F., & do Prado, A. F.: *An approach to the decomposition of business processes for execution in the cloud. In*: IEEE/ACS 11th *International Conference on Computer Systems and Applications* (AICCSA), pp. 470-477. IEEE (2014).
10. Sadiq, W., Sadiq, S., & Schulz, K.: *Model driven distribution of collaborative business processes*. *In*: IEEE *International Conference on Services Computing*, SCC'06, pp. 281-284. IEEE (2006).
11. Wodtke, D., Weißenfels, J., Weikum, G., & Dittrich, A. K.: *The Mentor project: Steps towards enterprise-wide workflow management*. *In*: *Twelfth International Conference on Data Engineering*, pp. 556-565. IEEE (1996).
12. Yu, Y., Ma, H., & Zhang, M.: *A genetic programming approach to distributed execution of data-intensive web service compositions*. *In*: *Australasian Computer Science Week Multiconference*. ACM (2016).
13. Caetano, A., Silva, A., Tribolet, J.: *Business Process Decomposition: An Approach Based on the Principle of Separation of Concerns*. *Enterprise Modelling and Information Systems Architectures*, vol. 5(1) (2010).
14. Fan, S., Hua, Z., Storey, V., Zhao, J.: *A process ontology based approach to easing semantic ambiguity in business process modeling*. *Data & Knowledge Engineering*, vol. 102, pp. 57–77 (2016).
15. Página inicial do Graphviz, http://www.graphviz.org/, acedido em 2017/06/29

Comparing the inaccessibility characteristics of CAN and CAN FD protocols

João de Sousa Alves and José Rufino

LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal,
fc41937@alunos.fc.ul.pt, jmrufino@ciencias.ulisboa.pt *

Abstract. The Controller Area Network (CAN) protocol, originally designed more than two decades ago, has been widely used in multiple domains, including industrial control, automotive and aerospace.
To overcome two important limitations of the original CAN protocol, low data transmission rates and small data frame payload sizes, a new specification, the CAN with Flexible Data rate (CAN FD), was issued.
This paper addresses timing and safety issues of the new specification, demonstrating that CAN FD, although showing an improvement of timeliness in the absence of errors, continues to exhibit (almost) the same shortcomings of the original protocol with respect to its use in the design of highly reliable real-time embedded systems.

Keywords: Real-time and embedded systems; Real-time communications; Reliability and safety; Non-intrusive system observability.

1 Introduction

The Controller Area Network (CAN) [6, 12] is a simple and robust protocol. Originally designed by Bosch [12] for automotive applications it is widely used in a number of other domains such as industrial control, medical devices, transportation and vehicular applications, as well as aerospace.

Nevertheless, a key issue was that the original CAN protocol design exhibit two important limitations [6, 12], which may preclude the application of CAN in more general domains. These limitations were: low data transmission rates (only up to 1 Mbps) and small data frame payload sizes (8 bytes, maximum).

An effort to overcome those limitations has resulted in the definition of a new protocol specification, known as CAN FD, the CAN with Flexible Data rate protocol [13]. CAN FD allows message encapsulation with payloads up to 64 bytes and switching to a faster bit signalling rate after network access, through the original CAN deterministic node/message arbitration scheme, has been decided. This paper studies and analyses the timing and safety characteristics of the new

* This work was partially supported by FCT, through funding of LaSIGE Research Unit, ref. UID/CEC/00408/2013, and FCT/CAMPUS FRANCE (PHC PESSOA), through the transnational cooperation project NORTH. This work integrates the activities of COST Action IC1402 - Runtime Verification beyond Monitoring (ARVI), supported by COST (European Cooperation in Science and Technology).

specification, under normal operating conditions and in the presence of errors, aiming to provide an answer to one fundamental question: did it worth it?

The paper is structured as follows. Section 2 briefly presents the CAN protocol, its frame formats and the evolution into CAN FD. Section 3 analyses the timing of both CAN and CAN FD protocols under normal operating conditions while Section 4 addressees their inaccessibility characteristics. A comparative evaluation of CAN and CAN FD timing properties is provided in Section 5. Sections 6 and 7 respectively address the related work and conclude the paper.

2 Controller Area Network

CAN is a multi-master network that uses a twisted pair cable as transmission medium [6, 12, 4]. The network maximum length depends on the data rate. The maximum values are: 40m @ 1 Mbps [4]. A single bit can be transmitted in the bus at a time. The signalling of a bit in the bus takes one of two values: *recessive*, also the state of an idle bus; *dominant*, which always overwrites a recessive value. This behaviour, together with the use of unique frame identifiers, is exploited for bus arbitration. A *carrier sense multi-access with deterministic collision resolution* policy is used [6, 12]. When several nodes compete for bus access, the node transmitting the frame with the lowest identifier always gets the bus. Frames that have lost arbitration or have been destroyed by errors are automatically scheduled for retransmission.

CAN FD: CAN with Flexible Data rate

The CAN FD protocol has two differentiating features: it enables the transmission of data frames with a payload up to 64 bytes; it introduces a secondary data rate at which part of the data frame can be transmitted. Limitations of current transceiver[1] technology do not allow secondary data rates higher than 8 Mbps [7]. For transmitting at an higher data rate, CAN FD [13] enables a mechanism of data rate switch at the BRS bit (Bit Rate Switch) of the data frame (Figure 1). A recessive BRS bit establishes a boundary separating:

- *arbitration-phase* - which uses the "normal" bit signalling rate, allowing the CAN deterministic arbitration scheme to operate properly;
- *data-phase* - enabling the use of a "higher" rate for bit signalling.

A third phase, also performed at the normal bit signalling rate, intended for frame acknowledgement, terminates the CAN FD data frame transmission.

Frame Formats

A *data frame* is a piece of encapsulated information disseminated on the network, which contains as payload a *message*, a user-level piece of information. A *remote*

[1] A transceiver (abbreviation of transmitter/receiver) is a physical (PHY) layer device that links a node to the network cabling infrastructure.

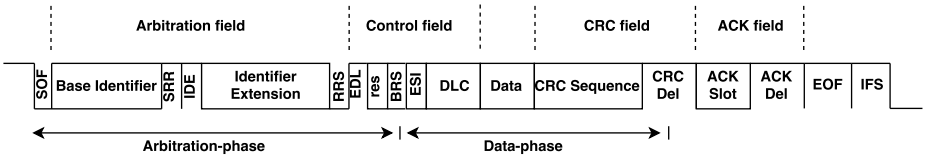


Fig. 1. CAN FD data frame structure and format.

frame has no payload and can be used to request the transmission of the data frame having the same identifier. Thus:

- *data frame* - includes the mandatory 11-bit frame's base identifier and possibly the optional 18-bit frame's identifier extension, as well as a number of control bits, which includes DLC, the 4-bit Data Length Code. DLC directly specifies the frame's payload length up to 8 bytes, the maximum payload size of the original CAN data frame. The remaining values encode the discrete lengths of 12, 16, 20, 24, 32, 48 and 64 byte in CAN FD data frames.
- *remote frame* - a CAN remote frame has a zero length payload and is identified by transmitting a recessive value in the RTR (Remote Transmission Request) bit. Since there is no advantage in encapsulating remote frames in CAN FD transmissions, it is replaced in CAN FD with a dominant value Remote Request Substitution (RRS) bit (Figure 1). This means, the remote frame format is not defined in CAN FD [13].

In the signalling of abnormal network operation incidents, the CAN protocol uses: *error frames*, for (global) error signalling; *overload frames*, to react to violations of the standard interframe spacing [6].

The signalling of an error detected during the data-phase of CAN FD implies switching first to the normal bit signalling rate which is then followed by the issuing of a standard CAN error frame. Thus, the CAN FD protocol does not formally defines nor uses own error/overload frame formats [13].

3 Analysing CAN and CAN FD Timings

This section establishes a set of easy-to-use expressions allowing the computation of the minimum and maximum normalised durations of a frame. The evaluation is independent of the actual bit signalling rate. To obtain the real duration of a frame, the normalised frame duration must be multiplied by the nominal bit time, $t_{bit} = \frac{1}{baud}$, being *baud* the nominal rate of bit signalling, on the bus. The normalised duration of a bit is represented by $\mathcal{T}_{bit}$, being $\mathcal{T}_{bit} = 1$ *bit-time*.

CAN Data and Remote frames

The main parameters that define the normalised duration of a frame are the frame format specification (base or extended) and the size of the payload field. With exception of the end-of-frame sequence, all the fields in a CAN frame are

subject to dynamic bit-stuffing coding. To establish a lower bound (^{lb}) for the duration of a data frame, we assume no bits are stuffed in the outgoing stream[2]:

$$\mathcal{T}_{data}^{lb} = (l_{fix} + l_{dlc} + l_{data} + l_{efs}) \,.\, \mathcal{T}_{bit} \tag{1}$$

where the meaning of the different length parameters in equation (1) is:

- l_{fix} is the length (in bits) of the fixed size fields subject to dynamic bit-stuffing. It includes the dominant one-bit *start-of-frame* (SOF) delimiter, the frame identifier and control bits, as well as the CRC sequence. The exact value depends on the CAN frame format specification (base or extended). However, it does not include l_{dlc}, the 4-bit DLC field;
- l_{data} is the length (in bits) of the payload field. It varies between 0 and 64, in 8 bit increments, being also subject to dynamic bit-stuffing;
- l_{efs} is the length (in bits) of the fixed form sequence, not subject to bit-stuffing, that ends every data or remote frame. It includes the CRC delimiter, the 2-bit acknowledgement field and the 7-bit *end-of-frame* delimiter.

To establish an upper bound (^{ub}) for the duration of a data frame, we assume that all the fields subject to dynamic bit-stuffing exhibit a pattern that leads to the maximum insertion of stuffed bits. Therefore:

$$\mathcal{T}_{data}^{ub} = \left(l_{fix} + l_{dlc} + l_{data} + \left(1 + \left\lfloor \frac{l_{fix} - l_{stuff} + l_{dlc} + l_{data}}{l_{stuff} - 1} \right\rfloor \right) + l_{efs} \right) . \mathcal{T}_{bit} \tag{2}$$

where $\lfloor \ \rfloor$ represents the *floor* function[3]; l_{stuff} represents the bit-stuffing width, i.e. the maximum number of consecutive bits of identical value that can be found in the outgoing stream, stuffed bits included. In the worst case, the first (recessive) stuffed bit is inserted in the outgoing stream immediately after the transmission of l_{stuff} initial dominant bits, starting with the SOF delimiter.

The minimum and maximum durations of a data frame can be derived by setting l_{data} to zero in equation (1) and by setting l_{data} to the maximum value (64 bits) in equation (2), respectively. For obtaining the minimum and maximum durations of a remote frame, l_{data} must be set to zero in equations (1) and (2).

CAN FD Data frames

The changes introduced by the CAN FD protocol [13] in the format of data frames, lead to the following updates to equations (1) and (2):

$$\mathcal{T}_{fddata}^{FD-lb} = (l_{fdix} - 1) \,.\, \mathcal{T}_{bit} + (1 + l_{dlc} + l_{data} + l_{fdcrc}) \,.\, \frac{\mathcal{T}_{bit}}{\sigma} + l_{efs} \,.\, \mathcal{T}_{bit} \tag{3}$$

[2] Equations (1) up to (8) do not account for the nominal three bit bus idle period, the interframe space, t_{IFS}, that usually precedes any data or remote frame and ends every CAN or CAN FD frame transmission (*intermission*).

[3] The *floor* function $\lfloor x \rfloor$ is defined as the greatest integer not greater than x.

$$\mathcal{T}_{fddata}^{FD-ub} = \left(l_{fdix} + \left\lfloor \frac{(l_{fdix}-1)-l_{stuff}}{l_{stuff}-1} \right\rfloor \right) . \mathcal{T}_{bit} + \left(1 + l_{dlc} + l_{data} + l_{fdcrc} + \left\lfloor \frac{(1+l_{dlc}+l_{data})}{l_{stuff}-1} \right\rfloor \right) . \frac{\mathcal{T}_{bit}}{\sigma} + l_{efs} . \mathcal{T}_{bit} \quad (4)$$

where $\sigma = \frac{baud\_high}{baud\_normal}$, is defined as the ratio between the higher and the normal data rates. The meaning of the new length parameters is as follows:

- l_{fdix} is the length (in bits) of fixed size fields subject to dynamic bit-stuffing transmitted at the lower data rate, which in CAN FD excludes the 4-bit DLC field and the full CRC sequence, as shown in Figure 1;
- l_{fdcrc} is the length (in bits) of the CRC sequence, which varies according to the size of the payload: 17 bit for payloads up to 16 byte; 21 bit for payloads longer than 16 byte. This parameter also includes the statically inserted fixed stuffed bits (FSB) in the CRC sequence, as illustrated in Figure 2.

The mandatory values taken by a relevant set of control bits, such as SRR, IDE, RRS, and EDL, in the CAN FD frame format (Figure 1), are not considered in the definition of Equation (4), which thus slightly overestimates the maximum number of bits that may be dynamically stuffed in the outgoing stream.

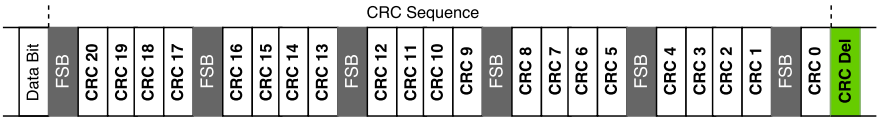


Fig. 2. CAN FD CRC 21 sequence with statically stuffed bits, as specified in [13].

Error and overload frames

Error and overload frames are not subject to bit-stuffing coding. The evaluation of their best (^{bc}) and worst case (^{wc}) durations is extremely simple. Equations (5) and (6) specify those durations for error frames:

$$\mathcal{T}_{error}^{bc} = (l_{flag} + l_{del}) . \mathcal{T}_{bit} \quad (5)$$

$$\mathcal{T}_{error}^{wc} = (2 . l_{flag} + l_{del}) . \mathcal{T}_{bit} \quad (6)$$

where l_{flag} and l_{del} are, respectively, the length in bits of the (dominant) six-bit *error flag* and of the recessive eight-bit *error delimiter*. Similar expressions can be established with regard to the evaluation of overload frame durations:

$$\mathcal{T}_{oload}^{bc} = (l_{flag} + l_{del}) . \mathcal{T}_{bit} \quad (7)$$

$$\mathcal{T}_{oload}^{wc} = (2 . l_{flag} + l_{del}) . \mathcal{T}_{bit} \quad (8)$$

4 Analysing CAN and CAN FD Accessibility Constraints

Data frame transfers are susceptible to errors due to bit corruption, e.g. by electromagnetic interference. Such errors may lead to periods of inaccessibility, a state where the network temporarily refrains from providing service, without that having to be necessarily considered a failure [18]. Networks have means of recovering from those situations. However, the recovery process takes time. So, in the meantime, the network is inaccessible.

This section addresses the inaccessibility characteristics of CAN FD, taking as basis previous studies on CAN [15, 19, 14]. In contrast with those studies, aiming comprehensive analyses, one address only errors affecting data frame transfers, the single protocol unity defined in CAN FD. Given CAN FD ability of transferring data at higher rates, susceptibility to errors increases.

Bounded and known message delivery latency, in the presence of faults, is fundamental to secure a real-time operation of CAN and CAN FD. Thus, this section is focused on a worst case analysis, being the exception situations where error signalling is issued in a pre-determined bit, making analytical computation of inaccessibility periods dependent only of data frame's payload size.

Bit Errors

In both protocols, transmitter-based error detection is achieved by listening to the bus while transmitting and comparing both streams on a bit-by-bit basis. A recessive level issued on the bus, by a given transmitter, can only be received back as dominant, without that being considered an error, in the following exceptional circumstances:

- inside the *arbitration field*, meaning the loss of the arbitration process;
- during the *acknowledgment slot*, meaning that at least one node has not detected, so far, any error in the current transmission;
- when a *passive* recipient[4] is transmitting an error flag.

All other situations are errors, signalled on the bus by starting the transmission of an error frame at the next bit slot. In CAN FD, this may imply switching back from the higher to the normal bit signalling rate. The corruption of the transmitted bit-stream cannot occur later than the transmission of the last bit of the *end-of-frame* delimiter. The corresponding inaccessibility period is obtained considering the transmission of data and error frames with the maximum size, for both CAN (Equation 9) and CAN FD (Equation 10) protocols:

$$\mathcal{T}^{wc}_{ina\leftarrow berr} = \mathcal{T}^{wc}_{data} + \mathcal{T}^{wc}_{error} + \mathcal{T}_{ifs} \tag{9}$$

CAN FD:
$$\mathcal{T}^{FD-wc}_{fdina\leftarrow berr} = \mathcal{T}^{FD-wc}_{fddata} + \mathcal{T}^{wc}_{error} + \mathcal{T}_{ifs} \tag{10}$$

[4] A node enters the passive state after being persistently affected by errors [6, 12, 13].

Bit-Stuffing Errors

The CAN and CAN FD protocols also use receiver-based error detection mechanisms. The first receiver-based error detection scheme to be analysed performs the monitoring of *bit-stuffing* violations. Whenever a receiving node monitors l_{stuff} consecutive bits of identical level, it dynamically and automatically deletes the next received (stuffed) bit. Under error free operation, the deleted bit presents a polarity opposite to the preceding ones; should this condition be violated, an error will be signalled on the bus, by starting the transmission of an error frame, at the next bit slot.

In CAN, the dynamic *bit-stuffing* coding scheme is used in the transmission of data and remote frames, up to the end of the 15-bit *CRC field*. A bit-stuffing error cannot occur after the reception of the last bit of the CRC field. Again, the worst case duration for an inaccessibility period occurs upon the transmission of a maximum size data frame, as specified by equation:

$$\mathcal{T}^{wc}_{ina\leftarrow stuff} = \mathcal{T}^{wc}_{data} - \mathcal{T}_{EFS} + \mathcal{T}^{wc}_{error} + \mathcal{T}_{ifs} \tag{11}$$

where $\mathcal{T}_{EFS} = l_{efs} \,.\, \mathcal{T}_{bit}$, accounts for the normalised duration of the fixed form sequence – not subject to bit-stuffing coding – that ends every data and, in the case of the CAN protocol, every remote frame.

In CAN FD, the *CRC sequence* is not subject to dynamic *bit-stuffing* coding and, as such, a bit-stuffing error can only be detected up to reception of the last bit of the payload and, consequently, signalled on the bus up to the reception of the first bit of the CRC sequence (Figure 1). Thus:

$$\underline{\text{CAN FD}}: \qquad \mathcal{T}^{FD-wc}_{fdina\leftarrow stuff} = \mathcal{T}^{FD-wc}_{fddata} - \mathcal{T}_{FD\_CRC} - \mathcal{T}_{EFS} + \mathcal{T}^{wc}_{error} + \mathcal{T}_{ifs} \tag{12}$$

where $\mathcal{T}_{FD\_CRC} = l_{fdcrc} \,.\, \mathcal{T}_{bit}/\sigma$, accounts for the normalised duration of the CRC sequence (Figure 2), fully transmitted, in the CAN FD protocol, at the secondary bit signalling rate.

Bit errors may occur in a way that they do not produce a violation of the bit-stuffing policy. In this case, the error will be detected slightly afterwards, either by *CRC checking* or through the detection of a frame format violation.

CRC Errors

After the reception of the *CRC sequence*, a node should do one of two things, depending on whether the CRC was good:

- if the CRC is correct, the node is obliged to change the bus level from recessive to dominant, during the *acknowledgement slot*;
- otherwise, the node does not modify the value of the *acknowledgement slot*, and signals the CRC error through the transmission of an error frame, that starts immediately after the *acknowledgement delimiter*. This allows to distinguish a situation where the CRC error is detected in all the nodes from the situation where such an error is detected only by a subset of the recipients.

The corresponding inaccessibility time is generically given by expression:

$$\mathcal{T}_{ina \leftarrow crc} = \mathcal{T}_{data} - \mathcal{T}_{EOF} + \mathcal{T}_{error} + \mathcal{T}_{ifs} \tag{13}$$

where $\mathcal{T}_{EOF}$ represents the normalised duration of the *end-of-frame* delimiter. The best and worst case bounds of equation (13) can be obtained considering the shortest and the longest data and error frame durations. An identical description is also valid for the CAN FD protocol. Thus:

CAN FD: $$\mathcal{T}_{fdina \leftarrow crc} = \mathcal{T}_{fddata} - \mathcal{T}_{EOF} + \mathcal{T}_{error} + \mathcal{T}_{ifs} \tag{14}$$

Acknowledgment Errors

Whenever a transmitter does not monitor a dominant level on the bus during the *acknowledgment slot*, it interprets that as an error, and the transmission of an error frame is started at the next bit slot. So, the duration of the corresponding network inaccessibility period is generically given by equation:

$$\mathcal{T}_{ina \leftarrow ack} = \mathcal{T}_{data} - \mathcal{T}_{EFS} + 2\,.\,\mathcal{T}_{bit} + \mathcal{T}_{error} + \mathcal{T}_{ifs} \tag{15}$$

The best and worst case inaccessibility bounds are obtained as usual, i.e. considering the minimum and the maximum durations for data and error frames. Specifically, in the CAN FD protocol:

CAN FD: $$\mathcal{T}_{fdina \leftarrow ack} = \mathcal{T}_{fddata} - \mathcal{T}_{EFS} + 2\,.\,\mathcal{T}_{bit} + \mathcal{T}_{error} + \mathcal{T}_{ifs} \tag{16}$$

In the presence of multiple errors, it may happen the transmitter monitors a faulty dominant level, at the *acknowledgement slot*. The transmitter cannot detect such an error. However, the lack of success in the transfer of the frame is signalled by the recipients that having detected a CRC error issue a negative acknowledgement, in the form of an error frame, at the bit slot immediately after the *acknowledgement delimiter*.

Form Errors

All the frames used by the CAN protocol obey to a few pre-defined formats. Data (and remote) frames have a fixed form end-sequence, where:

- the *CRC delimiter* and the *acknowledgement delimiter*, should always exhibit recessive values;
- the *end-of-frame delimiter*, consists of seven consecutive recessive bits.

An error in the frame ending sequence implies the transmission of an error frame. The longest period of inaccessibility caused by a form error is when it occurs while receiving the last but one bit of the *end-of-frame* delimiter, because a receiver monitoring a dominant level at the last bit of this delimiter does

not take that as a form error[5]. Thus, one will have for the CAN and CAN FD protocols, respectively:

$$\mathcal{T}^{wc}_{ina\leftarrow form} = \mathcal{T}^{wc}_{data} - \mathcal{T}_{bit} + \mathcal{T}^{wc}_{error} + \mathcal{T}_{ifs} \tag{17}$$

CAN FD: $$\mathcal{T}^{FD-wc}_{fdina\leftarrow form} = \mathcal{T}^{FD-wc}_{fddata} - \mathcal{T}_{bit} + \mathcal{T}^{wc}_{error} + \mathcal{T}_{ifs} \tag{18}$$

5 Evaluation and Discussion

This section aims to provide a comparative evaluation of CAN and CAN FD timing characteristics, both under normal operation and in the presence of errors.

Parameters	Frame field length (*bit*)							
CAN Parameters	l_{bid}	l_{eid}	l_{ctl}	l_{crc}	l_{fix}	l_{data}		
CAN Base format (2.0A)	11	-	4	15	30	≤ 64		
CAN Extended format (2.0B)	11	18	6	15	50	≤ 64		
CAN FD Parameters	l_{bid}	l_{eid}	l_{ctl}	l_{fdix}	l_{data}	l_{crc}	l_{fsb}	l_{fdcrc}
CAN FD Base format	11	-	7	18	≤ 128	17	5	22
	11	-	7	18	> 128	21	6	27
CAN FD Extended format	11	18	8	37	≤ 128	17	5	22
	11	18	8	37	> 128	21	6	27

l_{bid} and l_{eid}, are the base and extended identifier field length, respectively;
l_{crc} - CRC field length;
l_{ctl} - number of control bits, including the 1-bit SOF but excluding the 4-bit DLC field;
l_{fsb} - number of statically stuffed bits, in CAN FD.

Table 1. Field length parameters for different frame formats.

Frame	Normalised worst case frame durations			
	$\mathcal{T}$ (*bit-times*)		$\mathcal{T}$ (*bit-times*)	
	CAN Base (2.0A)	CAN Extended (2.0B)	CAN FD Base $\sigma = 8$	CAN FD Extended $\sigma = 8$
Data frame	132,0	157,0	115,1	138,1
Remote frame	52,0	77,0	-	-
Error frame	20,0	20,0	-	-
Overload frame	20,0	20,0	-	-

Table 2. Normalised CAN and CAN FD frame durations.

[5] This will be considered an *early reactive overload error*, whose analysis is out of the scope of this paper, but that was thoroughly studied in [15, 19, 14].

Fundamental parameters required for the assessment of both frame durations and periods of inaccessibility are established in Table1. The normalised worst case duration of the different frames, for the CAN protocol, and of data frames, for the CAN FD protocol, are inscribed in Table 2. The CAN FD protocol uses a speedup factor $\sigma = 8$ [7].

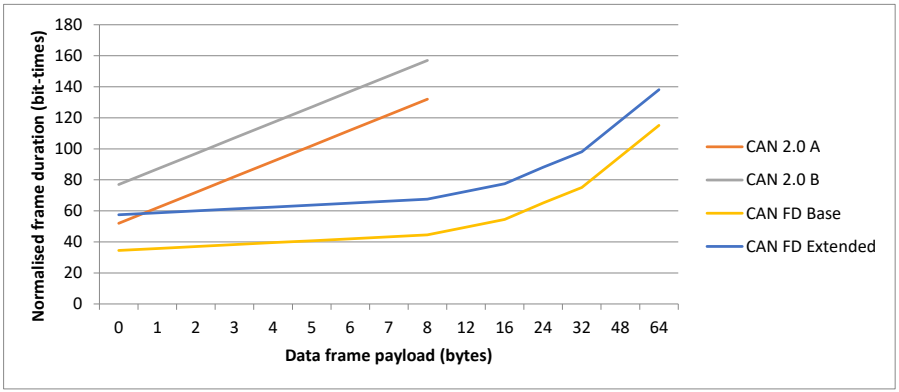


Fig. 3. Normalised CAN and CAN FD frame durations as a function of payload size.

The normalised data frame duration, for both protocols and for different payloads, is graphically represented in Figure 3. Again, the CAN FD protocol uses a speedup factor $\sigma = 8$ [7]. The timeliness of network operation, in the absence of errors, is thus summarily analysed in Figure 3.

The impact of CAN FD in the decrease of data frame transmission times and, therefore, in the system end to end message delivery latency, is evident in the graphic representation of Figure 3: for the same data frame format (e.g., 2.0B/extended) the data frame transmission times are always lower; for a zero payload the simple transmission of the CRC sequence (see, Figure 1) at a higher bit signalling rate is sufficient to lead to a lower data frame transmission time; as the data frame payload size increases, the difference between data frame transmission times, in the two protocols, becomes more significant.

Conversely, the improvements in network timing behaviour in the presence of faults are not particularly impressive, as shown by the numerical results of Table 3. Comparing the analytical expressions, already established for the CAN protocol [15, 19, 14], with those explicitly derived in this paper for the assessment of CAN FD inaccessibility periods, one conclude that, for most cases, the main difference concern the worst case duration of a data frame transmission.

Since the CAN FD protocol also specifies an increase in the maximum data frame payload size (from 8 byte in CAN to 64 byte in CAN FD), the positive effect of transferring the data frame payload at a higher bit signalling rate is counteracted by the need of transmitting a larger data frame payload. A speedup factor $\sigma \gg 8$ in the operation of the CAN FD protocol could mitigate such an effect, but that is not enabled by existing transceivers [7].

Error scenarios	Normalised worst case inaccessibility			
	$\mathcal{T}_{ina}$ (*bit-times*)		$\mathcal{T}_{fdina}$ (*bit-times*)	
	CAN Base (2.0A)	CAN Extended (2.0B)	CAN FD Base $\sigma = 8$	CAN FD Extended $\sigma = 8$
Bit errors	155,0	180,0	138,1	161,1
Bit-stuffing errors	145,0	170,0	124,8	147,8
CRC errors	148,0	173,0	131,1	154,1
Acknowledgement errors	147,0	172,0	130,1	153,1
Form errors	154,0	179,0	137,1	160,1

Table 3. Normalised CAN and CAN FD worst case inaccessibility bounds.

6 Related Work

A large body of research has been produced around the CAN protocol in the last two decades, addressing several timing related issues, namely: message schedulability analysis [17, 5]; performance of coding mechanisms [11] and behaviour in the presence of errors [3, 15], including their probability of occurrence [2]; integration of error behaviours in message schedulability analysis [9, 10].

Many of these works need now to be revisited and reformulated in the context of the CAN FD protocol. This paper is a step towards that direction, being the first contribution to a comprehensive analysis of CAN FD both in the absence and in the presence of errors.

Previous works on CAN FD are not numerous, being almost restricted to: the performance analysis under normal operating conditions [1]; a modelling and simulation study revisiting the automotive industry SAE benchmark in the context of CAN FD [16]; a detailed study of the improvements introduced in CAN FD with respect to error detection mechanisms [8].

7 Conclusion

This paper presented a comparative analysis of CAN and CAN FD protocols timing, both in the absence and in the presence of errors. Overall there is an improvement of timeliness characteristics but currently available CAN FD transceiver technology place a limit on such improvements. Nevertheless, in the absence of faults there is a significant improvement (in the order of 50%) in terms of the data transfers times for the same data payload length. In addition, the maximum data payload size has increased eight times, up to a maximum of 64 bytes, while maintaining a similar worst case message delivery latency bound.

In the presence of errors, given its correlation with the data frame duration upper bound, the improvements are less significant, exhibit lower numerical values and only a very modest enhancement is achieved.

On the other hand, the complexity of the protocol specification has increased. In a first analysis, some design decisions (e.g., the use of two alternative CRC

Assessing the Robustness of a Quadcopter's Flight Controller to Sensor Failures

Daniel Mendes[1], José Nunes[12], Sérgio Patrão[2], Naghmeh Ivaki[1], Pedro Amaro[2], and João Carlos Cunha[12]

[1] CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
[2] Instituto Politécnico de Coimbra, ISEC, DEIS, Portugal
dfmendes@student.dei.uc.pt, jnunes@isec.pt, smcpatrao@dei.uc.pt, naghmeh@dei.uc.pt, amaro@isec.pt, jcunha@isec.pt

Abstract. The rapidly growing range of Unmanned Aerial Vehicle (UAV) applications provides the basis for an increasing number of commercial and scientific solutions. One popular deployment of UAVs is on multirotor helicopters, namely quadcopters, which have been documented to be used in very diverse activities such as critical search and rescue, precision agriculture, crowd surveillance or hobby photography. Independently from their mission, the expanding use of quadcopters raises real concerns related to safety due to malfunction. The performance and safe operation of a quadcopter's flight controller heavily depends on the perception of the environment, and as such, it relies on a series of redundant and complementary sensing devices (e.g., accelerometer, gyroscope, barometer, and GPS). Data from these sensors, which are inherently inaccurate, are fused for correcting the deficiencies of the individual sensors and calculating the UAV's accurate position, orientation, and acceleration. In this paper, we aim to assess the robustness of the sensor fusion algorithm and flight controller in presence of faults affecting the sensors. To this end, we injected a number of faults in the sensors of a quadcopter using the ArduPilot flight controller platform. The quadcopter is programmed with several autonomous flight missions, and its behavior analyzed within a simulation environment.

Index Terms — Quadcopter, ArduPilot, Sensor Fusion, Fault Tolerance, Fault Injection

1 Introduction

Among all UAVs, quadcopters (or, in general, multirotor helicopters) are increasingly receiving attention in research, mainly due to their wide range of applications, their potential to perform tasks that may threat humans' safety, their simplicity in terms of mechanical design, low price, and last but most importantly, for their flexibility in movement [7, 15]. The wide range of applications for quadcopters with autonomous flight capabilities prompted a large number of commercial and academic proposals. Despite their flying ability and broad applicability, the dependable control of quadcopters has been considered as a challenge and is receiving attention from researchers of different areas, mainly due to the fact that it consists of several potentially faulty (e.g., flight controller software) and dangerous components (e.g., high-speed motors and propellers).

A quadcopter is a kind of helicopter with four rotors that make it very agile to attain the full range of motion [13] (capability of hovering, horizontal flight, vertical take-off and landing). To fly a quadcopter, a series of redundant and complementary sensing devices (e.g., accelerometer, gyroscope, compass, barometer, magnetometer, and GPS) together with

data filtering solutions (e.g., Extended Kalman Filter (EKF)) are used by a flight controller (e.g., ArduCopter), to continuously compute its position and orientation. Data from the sensors, which are inherently inaccurate, are combined for correcting each other's deficiencies and computing the flight control commands. However, ensuring the robust operation of a quadcopter is challenging because all sensors and filtering solutions are faulty and subject to noise and environmental disturbances.

We can find several works in the literature trying to address the challenges associated with the robust and dependable control of a quadcopter. A fault diagnosis system is developed in [6] to detect and isolate the sensors' faults. In [16], the authors design a flight control system that is tolerant to rotor failure. Similarly, in [17] the authors design a controller that hovers around a given position despite losing the control of a single, two opposing, or three propellers. An adaptive control of quadcopter based on trajectory tracking is proposed in [5, 23] to increase robustness against parametric uncertainties, component failure, physical damage, or sensors faults.

Our work is concerned with the sensors erroneous readings and their impact on the robustness of the quadcopter's flight controller. We use a fault injection technique to assess the robustness of a commonly used flight controller, namely ArduCopter [4], during unmanned missions, in presence of sensors faults. Each mission is executed in a fault-free scenario, called as Gold Run (GR), to measure the vehicle's deviation from the reference paths defined in the missions, in normal circumstances. The results of GRs are used as oracle to evaluate the vehicle's deviation after injection of faults. The injected faults affect the value of data read from sensors immediately before being fed to the flight controller.

The rest of this paper is organized as follows. Section 2 describes how quadcopters sensor fusion and flight control work. Section 3 describes the architecture of the System Under Assessment (SUA). The methodology used to assess the robustness of the SUA is described in Section 4. The results obtained are presented, analyzed and discussed in Section 5. Finally, Section 6 concludes the paper.

2 Quadcopter flight control and sensor fusion

Quadcopters are underactuated systems with six degrees of freedom (three translational and three rotational) that are achieved by actuating four DC motors [9]. Their movements are controlled by increasing or decreasing the power delivered to the motors, thus changing their rotational speeds with required precision for maneuver and flight. Controlling Quadcopters is considered as a complex task that is difficult to achieve without an automatic flight controller.

An automatic flight controller combines various sensors' and GPS's data with pilot commands and current state of the quadcopter (position, velocity and attitude) to calculate each motor input power so that the vehicle desired movement is achieved. The controller's main function is to recursively minimize the difference between the measured and desired state of the UAV by adjusting the power delivered to the motors based on the received sensors' data and pilot command. At the heart of the flight controllers, there are sensor fusion algorithms, which are used to correlate data from different sensors to estimate the vehicle's state. Thus, robustness of the flight controller to sensors' inherent inaccuracy and noise, which is crucial for correct estimation of vehicle's state and its steady flight behavior, mainly depends on the functionality of these sensor fusion algorithms.

One of the most commonly employed sensor fusion algorithms is the Kalman Filter (KF), first presented by R. E. Kalman in 1960 [12]. KF uses linear modeling to estimate the state of a system from indirect and noisy sensor data [22]. The Extended Kalman Filter (EKF),

which has been subject of extensive research and real applications specially in navigation of autonomous vehicles, is an extension of the Kalman Filter for nonlinear systems [11]. Based on system's current state, the EKF is able to linearize nonlinear state functions.

EKF operation may be characterized by two sets of equations: time update equations and measurement update equations. Time update equations compute the state and error covariance estimates from the previous time step to obtain the *a priori* estimates. Measurement update equations can be considered as corrector equations, due to the fact that they are responsible for correcting the predicted estimates with a new measurement, to obtain the *a posteriori* state and error covariance estimates [22].

The correction is done by weighing the difference between the actual and the predicted measurement, called innovation. Associated with the observation measurements, there is a noise covariance matrix that indicates the noise variance in the measurements. This matrix is a tunning parameter of the filter. When it is close to zero, the actual measurements have higher weight in the state estimation, while the predicted measurements are less trusted. On the other hand, when the *a priori* error covariance estimate approaches zero, the actual measurements are less trusted, while the predicted measurements have higher weight in the state estimation. Although this adaptive behavior of EKF provides the controller with a level of robustness against sensors' noise and interferences and the ability to recover from wrong estimates, it is far from being enough to guarantee a safe flight in presence of sensors malfunction or failure.

A sensor malfunction or failure may cause the quadcopter to fly erratically or in a wrong direction, depending on a number of parameters that relate to the flight controller (e.g., RTL speed, Battery Failsafe Enable), the EKF (e.g., Magnetometer measurement noise), the inaccuracy of sensors data (e.g., Accelerometer error threshold), and the environment in which the quadcopter is flying (e.g., Wind velocity). A sensor malfunction or failure can be temporary or permanent, and its inaccurate data might be simply discarded by the EKF or it might be used for navigation of the vehicle. Moreover, the sensors' role in controlling the vehicle differ from each other, therefore, the malfunction of each sensor may differently influence the vehicle behavior.

3 Quadcopter System Structure

The quadcopter used in our experiments uses an on-board computing system, a Raspberry Pi [21], and a Navio 2 control board developed by Emlid [2]. A simplified view of these two components of the quadcopter's control system is presented in Figure 1.

The Raspberry Pi that serves as the brain of this system includes a lightweight operating system, called `Raspbian`, which is built based on Linux Debian [8], and an open-source control software referred to as `ArduCopter` [4].

The role of the Navio 2 board is to serve as an intermediary between the ArduCopter and the external physical components, like sensors, radio communication, and motors. For instance, it uses a `PWM generator` to encode messages coming from ArduCopter into pulsing signals allowing to control the quadcopter's motors. It also implements a `PPM decoder` to decode Pulse Position Modulation (PPM) signals from a Radio Control (RC) receiver, so it can be understood by the autopilot software. In addition to this, Navio 2 has the hardware, including two Measurement Units (`IMUs`), `barometer` and `GPS` module, needed by the ArduCopter to control the quadcopter's flight.

The ArduCopter has two core functionalities: sensor fusion and flight control. It implements the Extended Kalman Filter (`EKF`) to fuse data received from sensors and GPS. The EKF's outcome plus the information (in case there is any) coming from the PPM decoder

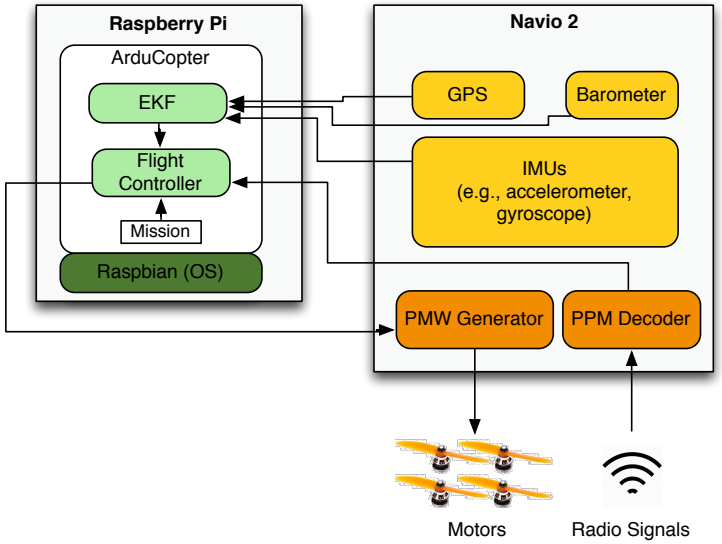


Fig. 1: General view of the quadcopter 's system.

(e.g., pilot commands) or the information of a predefined mission stored locally are used by the flight controller to calculate the position, altitude, and orientation of the quadcopter and navigate in real-time.

The EKF filtering process can be divided into two main phases: State Prediction and Measurement Fusion. In the State Prediction phase, inertial navigation equations are used to calculate the change in orientation, velocity and position since the last update. For this purpose, gyroscope and accelerometer data are used. Then, the filter projects the state and covariance estimates from the previous time step to the current time step, also known as *a priori* estimates. In the Measurement Fusion phase, the filter updates the *a priori* state and covariance estimates using measurements from the sensors. The filter processes data from magnetometer, GPS and barometer to find the difference between the measurements predicted by the filter and the sensors' measurements, called innovation. The innovation is then multiplied by the Kalman gain to calculate the corrections needed to be applied to the predicted estimates, to find the *a posteriori* estimates. The recursive nature of the filter comes from the fact that after each time step the process is repeated with the previous *a posteriori* estimates being used to predict the new *a priori* estimates. A complementary filter is also used to correct time delays caused by different sampling rates of the different sensors used by the EKF [14].

The default sensor fusion algorithm for navigation control running on ArduCopter is, at the time of writing this paper, a 24-state Extended Kalman Filter [20]. The EKF state vector is composed of the following states: attitude, velocity, position, gyroscope bias offset, gyroscope scale factors, Z-axis acceleration bias, Earth magnetic field, Body magnetic field, and wind velocity. Although the filter outputs attitude in quaternion format, it does not attempt to estimate it directly. Instead, it estimates an error rotation vector and applies that correction to the quaternion from the inertial navigation equations implemented in the State Prediction phase [19]. The 24-state EKF is also ready to use laser range finder, optical flow and airspeed sensors as observations, despite the fact that only magnetometer, GPS and barometer are available as part of the flight controller hardware used in this work.

4 Robustness Assessment Methodology

To assess the dependability of an embedded system and the fault tolerance mechanisms in place, a very common approach is to inject faults in the system and then monitor its behavior. This is known as Fault Injection (FI) [10]. There are two types of faults that can be injected: hardware faults and software faults [18]. The former ones includes memory, bus, and CPU faults, which can be implemented either physically, by using techniques such as radiation induced and power supply disturbances, or emulated by software. The later includes bugs that affect software components, mainly originated throughout the software development phase. In this work, we assess the robustness of the ArduCopter open source flight controller, by injecting hardware faults (erroneous sensor data) emulated by software. When compared to physical approach, software emulation has higher precision and better controllability with a lower cost. The major challenges involved with the emulation of hardware faults are the representativeness of the injected faults and their intrusiveness. In this section, we present the details of the experiments performed, the faults injected, and the methods used for analysis.

4.1 Experimental Setup

Although the goal is to use the Raspberry Pi / Navio-2 based quadcopter, for the experiments in this paper we used a simulator, namely the Software In The Loop (SITL) from the ArduPilot repository [3]. SITL simulates the physics of several UAVs including quadcopters, and uses the same controller software that runs on the embedded devices, namely the Raspberry Pi. The simulated devices are built on top of ArduPilot, both running on a PC without any extra hardware. In fact, when flying a quadcopter in SITL, the sensor data comes from a flight dynamics model defined in the SITL simulator. This allows us to simply test the behavior of the quadcopter without using a real device. An overview of SITL with the rest of the experimental setup is shown in Figure 2.

To precisely mimic a real system, SITL models all necessary hardware (e.g., motors, sensors, GPS) and allows to configure a large set of parameters related to noise in sensors and GPS and environmental disturbances (e.g., wind speed and direction). In our experiments, we used the default noise-free setting of SITL to make sure that our results are not influenced by other noises and environmental disturbances.

As explained in Section 3, ArduCopter, our System Under Assessment (SUA), is composed of an Extended Kalman Filter (EKF) and a flight controller. It reads the flight mission file stored locally and navigates the simulated quadcopter by sending the necessary commands. The detailed information of each flight, from beginning to the end of the mission, including the data read from each sensor and GPS, the vehicle's position and the output of EKF, is logged internally by the SUA. We connect to the SUA through a Ground Control Station (GCS), namely MAVProxy, using a TCP/IP connection, to download all the SUA's internal logs after all tests have been successfully finished. In order to automate the execution of the experiments, we use a remote commander implemented in Python, running on DroneKit [1], and connected to the GCS using a UDP/IP connection, to send the necessary signals to the SUA.

In order to inject the faults, the source code of the ArduCopter is instrumented with a fault injection module consisting of a few hookup functions allowing to change the data read from sensors before being fed to the SUA, in particular to its EKF. The information of the faults to be injected is prepared previously to the missions. Figure 2 shows the experimental setup and the location of the injected faults.

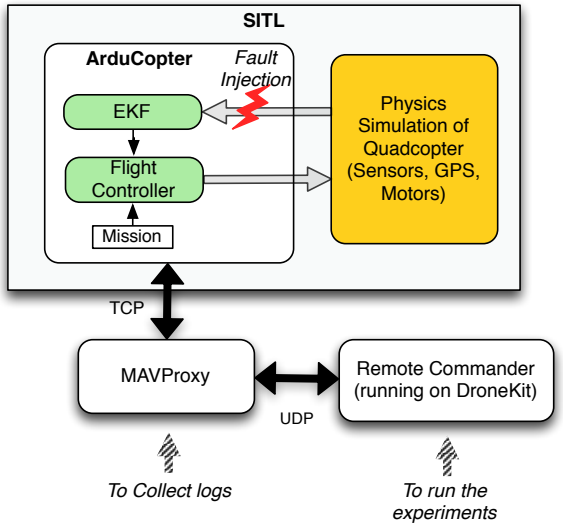


Fig. 2: High level block diagram of our experimental setup.

4.2 Quadcopter's Missions

We used the following three simple missions representing the basic trajectories involved in most of the flights:

- **Loiter** mission: consists of i) takeoff (1.5 m) to the first waypoint; ii) staying at the same position (for about 20 seconds); and iii) land.
- **Vertical** mission: consists of i) takeoff (1.5 m) to the first waypoint; ii) rising (0.5 meter) to the second waypoint; iii) returning to the first waypoint; iv) repeating ii and iii ; and v) land.
- **Horizontal** mission: consists of i) takeoff (1.5 m) to the first waypoint; ii) going forward (1.5 m) to the second waypoint; iii) returning back to the first waypoint; iv) repeating ii and iii; and v) land.

The missions take respectively 28, 24 and 36 seconds to execute (excluding arming the motors, i.e., applying power to the motors, which takes about 30 seconds). Figure 3 presents a 3D plot of these missions, with the waypoints (denoted by asterisks), reference trajectories in red, and the simulated trajectories in blue.

4.3 Fault Model

As already explained earlier in this Section, we have injected faults in the SUA by changing the readings from the sensors, namely the quadcopter's IMUs (gyroscope, accelerometer, magnetometer) and barometer sensors (temperature and pressure). We have not injected faults in the GPS readings, as we rely on the GPS for tracking the quadcopter's position.

The fault model defined in this work has four dimensions: trigger, location, duration, and type. The **fault trigger** is the moment when the fault is activated. We have used a simple time-triggered fault injection, meaning that the fault is injected after 40 seconds from the beginning of each flight (including arming the motors and executing the mission). In this work we have not explored other injection moments, such as a change in trajectory, since the other dimensions are more important and require our attention for now.

The **fault location** refers to the sensors that are targeted for fault injection:

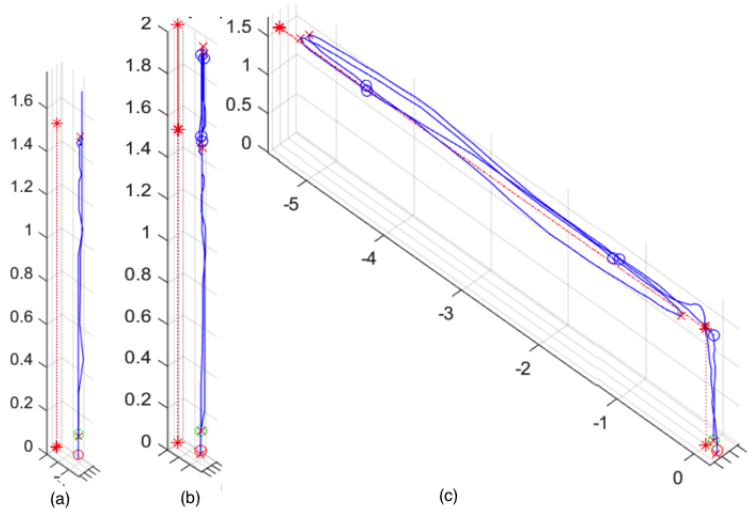


Fig. 3: Simulated missions: a) loiter, b) vertical, and c) horizontal (distances in meter).

- **Accelerometer** measures accelerations and can detect small changes in velocity and position (through integration), allowing the vehicle to be stable and maintain balance;
- **Magnetometer** measures magnetic fields and can be used as a compass, providing information about the orientation of the vehicle with respect to the cardinal direction (NESW plane);
- **Gyroscope** measures changes in orientation or rotational velocity and allow the vehicle to detect sudden changes in orientation. It needs to start from a known orientation, usually provided by a magnetometer;
- **Pressure sensor** measures air pressure (barometric pressure) and provides a very good indicator of the vehicle altitude. Pressures is directly proportional to the absolute temperature, which makes a temperature probe crucial;
- **Temperature sensor**: measures temperature and provides valuable information for the correct measurement of the barometric pressure.

Regarding the **fault duration**, faults may be classified as temporary or permanent. A temporary fault lasts for a specified amount of time, whereas a permanent fault affecting a sensor produces an erroneous output that lasts until the end of the mission. We have defined faults with duration between 0.05s and 4s, as well as permanent.

Finally, the **fault type** represents the way a sensor may fail. In this work, the faults are modeled as follows: i) stuck-at, the value stays at zero or saturation (maximum admissible value - specific to each sensor); ii) freeze, the value keeps the present reading during the fault injection period; iii) altered, the value is changed to a number inside or outside the valid range (doubled or halved). According to these models, we defined 6 types of faults as follows:

- **Zero**: the value of the sensor is set to zero;
- **Half**: the value of the sensor is set to half of the reading;
- **Freeze**: the value of the sensor remains equal to the previous readings;
- **Double (dValue)**: the value of the sensor is doubled;
- **Max**: the value of the sensor is set to its maximum;
- **Double Max (dMax)**: the value of the sensor is set to double of the maximum value. This means that an out-of-range value is produced by the sensor.

Table 1 presents the maximum value that each sensor is able to read. It is worth mentioning that if a sensor reading produces more than one single value, as is the case of the accelerometer (a three element vector with the vehicle accelerations in X, Y and Z axis), all the values are affected in the same manner by the injected fault.

Table 1: Maximum admissible value for each sensor.

Sensors	**Maximum Value**
Magnetometer	$49000mG$
Accelerometer	$16m/s^2$
Gyroscope	$34.89rad/s$
Pressure sensor	$120000Pa$
Temperature sensor	85^o

Only one fault is injected at a time. Each experiment consists of starting the mission, injecting the fault, and collecting the results. Afterwards, and before the next fault-injection experiment, the SUA is automatically reset in order to avoid the accumulation of fault effects.

4.4 Experimental Data Analysis

The SUA is a feedback control system, having as inputs the data from the sensors, having as outputs the control actions to the motors, receiving reference points to accomplish the mission, and containing an internal state. As already explained, in our experiments the physical system is substituted by a software simulation, which receives the control actions (to actuate in the motors) and produces the sensor's values. The solution devised to analyze the robustness of the system regarding the injected faults was to evaluate the behavior of the whole system (i.e., the controller plus simulator), namely by observing the three-dimensional (x,y,z) position of the quadcopter. These Cartesian coordinates are obtained from the GPS values generated by the simulator.

On a regular mission, i.e. without injected faults and without external disturbances, the quadcopter accomplishes its mission by following a predefined path, though joining consecutive points, known as waypoints (see in Figure 3 the red stars and the red lines that connect them). Due to some indeterminism and to the control algorithms characteristics, this path is not followed in a precise straight line, but rather deviates and approaches the reference path along the way (see in Figure 3 the blue lines).

We have thus determined that the correct behavior of the quadcopter is when its trajectory is consistently within a maximum distance from the reference path, meaning that it does not fly out of a corridor, or tube, as depicted in Figure 4. The radius of this tube should be large enough to contain the path of a fault-free mission. We have thus run the three missions (loiter, vertical and horizontal) 10 times each to observe the normal behavior of the quadcopter. We then collected the logs of the missions, converted the GPS values into 3D Cartesian values, and calculated the maximum distance between the trajectory and the reference path: 0.31m. Adding a 10% margin, we determined that if the quadcopter remains within a maximum distance of 0.341m from the reference path (i.e. inside a tube with a radius of 0,341m), then its behavior is considered as **normal**.

If, on the contrary, due to a fault, the path of the quadcopter goes outside this margin, then a failure has occurred. However, we consider there is a *safe* distance from the predefined trajectory that should be regarded in order not to cause injuries. If the quadcopter runs outside the *normal* margin, but inside the *safe* margin, there was a **minor failure**. Running outside the *safe* margin incurs in a **major failure**. We have arbitrarily defined the safe

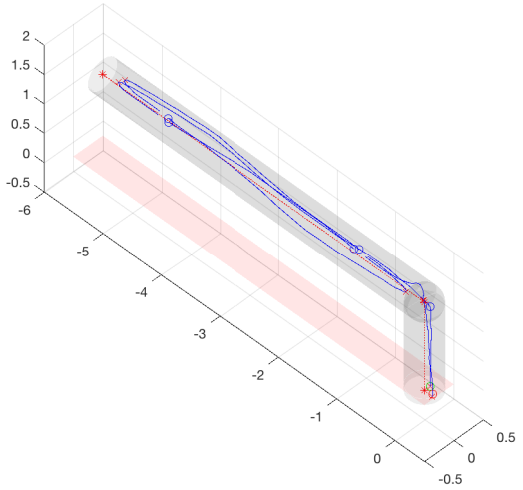


Fig. 4: Corridor followed in a fault-free mission (tube in gray).

margin as double the maximum distance the quadcopter runs from the reference path, i.e. 0.62m.

If the quadcopter is not able to finish its mission, due to hitting the ground or not returning to the path within a timeout of 60 seconds, then we consider a **crash** occurred.

For each experiment we injected a single fault and collected the corresponding log file. Then this file was fed to a Matlab script to calculate its maximum distance from the expected trajectory. According to the calculated maximum distance, we classified the experiments as resumed in Table 2.

Table 2: Experiment Outcomes.

Outcome	**Description**
Normal	The deviation is not noticeable $deviation \leq normal\ margin$
Minor Failure	The deviation is noticeable, but still acceptable $normal\ margin < deviation \leq safe\ margin$
Major Failure	The deviation is not acceptable, but the vehicle still manages to accomplish the mission $deviation > safe\ margin$
Crash	The quadcopter hits the ground or fails to complete the mission

5 Results, analysis, and discussion

In this section, we present and analyze the results obtained from the experiments.

5.1 Preliminary Experiments

We started our experiments by injecting some random faults, with different durations and in different missions. From this set of experiments, and after analyzing the results, we had two simple and general observations:

– The faults affecting the Barometer (Pressure and Temperature sensors) had no noticeable impact on the behavior of the SUA, even after increasing the duration of the faults;
– By running the same fault-injection experiment multiple times, we got the same outcomes.

This way we decided to exclude the Pressure and Temperature sensors from the analysis, and decided that a single run of each fault-injection experiment would be enough. For illustration purposes, we present in Figure 5 a plot ot the SUA trajectories with three selected faults that led to a) a minor failure, b) a major failure, and c) a crash.

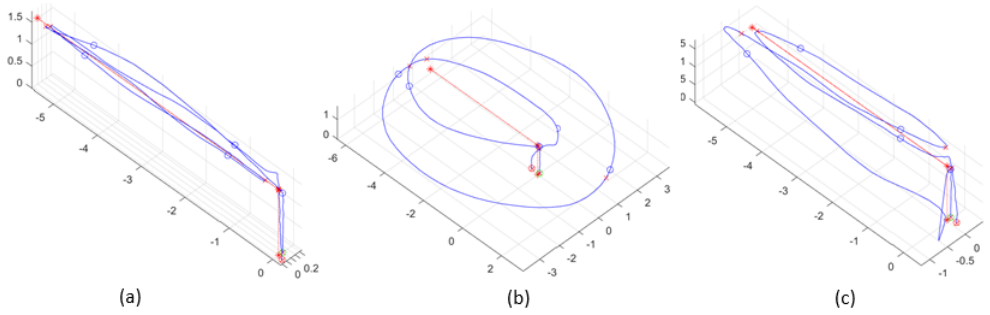


Fig. 5: Plots of SUA failures: a) minor, b) major, and c) crash).

5.2 2-seconds Faults

We continued then to run the experiments in a systematic way, and injected 2-seconds faults of each type, targeting all sensors, and affecting all the missions. Table 3 presents the maximum deviations, in meters, between the reference and each simulated trajectory. It started to be clear from these results that faults which push the sensor value to its maximum (*Max* and *Double Max*) have the highest impact, while zeroing or freezing the sensor values have no impact on the SUA behavior. We also observed that the faults injected in the accelerometer show the highest impact on the SUA behavior, while the faults in the magnetometer show the lowest impact. In fact, the impact of a fault targeting this sensor is only noticeable when the quadcopter is executing a horizontal movement.

Table 3: Impact of 2-seconds faults of all types in all sensors

	Magnetometer			Accelerometer			Gyroscope		
	Loiter	Horizontal	Vertical	Loiter	Horizontal	Vertical	Loiter	Horizontal	Vertical
Zero	0.31	0.25	0.15	0.31	0.25	0.15	0.31	0.25	0.15
Half	0.31	0.24	0.15	0.86	0.85	0.76	0.31	0.25	0.15
Freeze	0.31	0.25	0.15	0.31	0.25	0.15	0.31	0.25	0.25
dValue	0.31	0.25	0.18	0.31	0.73	2.14	0.31	0.24	0.15
Max	0.30	Crash	0.18	15.26	Crash	14.81	Crash	Crash	Crash
dMax	0.30	Crash	0.18	15.38	Crash	15.04	Crash	Crash	Crash

5.3 Extending fault duration

Starting from the previous results, we have extended the fault injection experiments by using other fault durations, above and below 2 seconds. We have however not injected faults when the results were known *a priori*. For example, when a crash occurs with a 2-seconds fault, it is not necessary to inject the same fault with a higher duration; similarly, if a 2-seconds fault has no effect, it is not necessary to inject the same fault with lower duration. We have

also injected permanent faults, i.e., faults that remain active until the end of the mission. In total we have injected 152 faults.

Fault Type vs. Location & Duration The overall results assuming the aggregate of all the missions, is presented in Table 4, where fault type versus location and duration is more visible. From these results we observe that:

- the Accelerometer is the most sensitive sensor, where the majority of the interferences, with a duration higher than 1 second lead to a major failure;
- a Magnetometer fault does not compromise the system, except when it saturates the outputs to the maximum value;
- a sensor freeze is normally tolerated by the system, being the Gyroscope the only exception, when in the presence of a permanent fault.

Table 4: General outcomes: fault type vs. fault location & duration.

	Accelerometer					Gyroscope					Magnetometer				
	0.05	1	2	4	P	0.05	1	2	4	P	0.05	1	2	4	P
dMax	✧	✦	●	●	●	✧	●	●	●	●	✓	●	●	●	●
Max	✧	✦	●	●	●	✓	●	●	●	●	✓	●	●	●	●
dValue	✓	✓	✦	✦	●	✓	✓	✓	✧	✧	✓	✓	✓	✓	✓
Freeze	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	✓	✓	✓	✓	✓
Half	✓	✦	✦	✦	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Zero	✓	✓	✓	✦	●	✓	✓	✓	✓	●	✓	✓	✓	✓	✓

✓ no effect ✧ minor ✦ major ● crash

Mission vs. Fault duration & location Table 5 relates the missions with the fault duration and location, for the worst case fault type. By closely analyzing the 3 different missions, we come to the following conclusions:

- faults with a very short duration are well tolerated by the system, leading at most to minor failures;
- the Horizontal mission is the most sensitive to higher duration faults, of more than 1 second;
- faults in the Magnetometer do not affect the SUA when executing both the vertical and loiter movements.

Table 5: General outcomes: mission vs. fault duration & location.

	0.05 seconds			1 second			2 seconds			4 seconds			Permanent		
	Acc	Gyr	Mag	Acc	Gyr	Mag	Acc	Gyr	Mag	Acc	Gyr	Mag	Acc	Gyr	Mag
Loiter	✧	✧	✓	✦	●	✓	✦	●	✓	✦	●	✓	●	●	✓
Vertical	✓	✧	✓	✦	●	✓	✦	●	✓	✦	●	✓	●	●	✓
Horizontal	✧	✓	✓	✦	●	●	●	●	●	●	●	●	●	●	●

✓ no effect ✧ minor ✦ major ● crash

6 Conclusion

In this paper, we explored the robustness of a quadcopter based on the popular ArduCopter flight controller, to sensor failures. It is well known that sensors are inherently unreliable, and thus, the control systems of intelligent vehicles make use of sensor fusion algorithms, like Kalman Filters, to estimate the position, velocity and attitude of these systems, despite any

Approximate Duplicate Elimination using State-Of-The-Art Tools: a Comparison

João L. M. Pereira and Helena Galhardas

INESC-ID and Instituto Superior Técnico, Universidade de Lisboa
{joaoplmpereira,helena.galhardas}@tecnico.ulisboa.pt

Abstract. Successfully analyzing data is a challenge for the scientific community and companies since raw data usually contains several data quality problems that influence data-based decisions. The presence of records that refer to the same entity in the real world is one of the most common data quality problems that greatly impact data analytics. In order to guarantee a unique and proper representation record for each entity, an Approximate Duplicate Elimination procedure has to be performed. It identifies records that refer to the same entity and then generates a unique record representation for each entity. Distinct commercial and research tools have been developed to carry out Approximate Duplicate Elimination. However, there is no systematic comparison of tools regarding their support for Approximate Duplicate Elimination.
This paper aims at providing a comparison of three different types of tools that can perform the Approximate Duplicate Elimination task, namely Pentaho Data Integration, CLEENEX, and OpenRefine. For this purpose, in order to support an experimental evaluation, we generated a synthetic dataset with errors, we specified the same Approximate Duplicate Elimination task for each tool, and then we evaluated these three tools using different criteria. With this comparison, we expect that users are able to choose the most appropriate tool or a combination of tools that better suit a specific Approximate Duplicate Elimination task.

Keywords: Approximate Duplicate Elimination, Record Linkage, Data Cleaning, Pentaho Data Integration, OpenRefine.

1 Introduction

Poor data quality can have disastrous consequences on data-based decisions. A very common data quality problem is the existence of records that are not exact duplicates but refer to the same entity in the real world. For instance, the presence of more than one Electronic Health Record referring to the same patient that contains partial or erroneous information in a database can lead to bad decisions about a certain medical treatment. The data cleaning task that aims at finding those records that represent the same real world entity and then choosing a unique representation for them, is called Approximate Duplicate Elimination, also known as Record Linkage [6].

Automatic procedures to execute an Approximate Duplicate Elimination process can be implemented using a programming language such as C, Java, or Python, or through specialized tools. Specialized commercial and research tools support different implementations of Approximate Duplicate Elimination. For instance, ETL (Extraction, Transformation and Loading) tools provide several data operators that support data processing tasks. Most commercial ETL tools such as Informatica PowerCenter[1], IBM Data Integration[2], Talend[3], and Pentaho Data Integration (PDI)[4] can support the Approximate Duplicate Elimination task. Nevertheless, only PDI provides a free version with enough data operators to support this task. Alternative tools to perform Approximate Duplicate Elimination are data cleaning tools. Examples of data cleaning tools are Trillium Quality[5], Febrl [5], LLUNATIC [11], NADEEF [7], and CLEENEX[6] based on Ajax [8][9]. CLEENEX is the only free data cleaning tool that supports data operators for Approximate Duplicate Elimination while providing a Graphical User Interface. Finally, the data wrangling tools, Trifacta Wrangler[7] based on the research tool Data Wrangler [12] and OpenRefine[8], provide an interactive process for transforming and mapping data through data operations. However, OpenRefine is the only data wrangling tool that supports Approximate Duplicate Elimination tasks.

Recently, [1] and [2] evaluated data cleaning tools but the authors of [1] focus on error detection tasks and the authors of [2] on data quality constraints. None of these works evaluated tools in what concerns their support for the Approximate Duplicate Elimination task.

The objective of this work is to perform an analytical and experimental evaluation of three distinct types of tools that can perform the Approximate Duplicate Elimination task: PDI, CLEENEX, and OpenRefine. The evaluation process includes: (i) the controlled generation of errors for a synthetic dataset; (ii) the specification of a data cleaning program to be executed by each tool; and (iii) the evaluation of the three tools in order to assess their support and performance to carry out the Approximate Duplicate Elimination task.

This document is organized as follows. In Section 2, we describe a detailed motivating example for this work. In Section 3, we introduce the three tools compared in this work, namely PDI in Section 3.1, CLEENEX in Section 3.2, and OpenRefine in Section 3.3. Then, we describe the analysis and the experimental evaluation of these tools and report the results obtained in Section 4. Finally, in Section 5, we review the main conclusions of this work and a possible future line of work in the comparison of tools for data cleaning tasks.

[1] https://www.informatica.com/products/data-integration/powercenter.html
[2] https://www.ibm.com/analytics/us/en/technology/data-integration/
[3] https://www.talend.com/
[4] http://community.pentaho.com/projects/data-integration/
[5] https://www.trilliumsoftware.com/products/tss/data-quality
[6] http://web.tecnico.ulisboa.pt/˜ist164790/cleenex/
[7] https://www.trifacta.com/
[8] http://openrefine.org/

Fig. 1: Transforming a table containing authors last name by publication into a clean table containing author last names.

(a) Initial table.

PublicationAuthors

Pub ID	**Authors**
1	Kanamori, Heiken
2	Kaanamori
3	Reeken, Kanamori, others
4	Knamoqri, Yothers

(b) Table with a final representation for each author.

CleanAuthors

Author ID	**Name**
1	Kanamori
2	Heiken
3	Reeken
4	Yothers

(c) Table containing a list of author names.

Authors

Author ID	**Name**
1	Kanamori
2	Heiken
3	Kaanamori
4	Reeken
5	Kanamori
6	others
7	Knamoqri
8	Yothers

(d) Pairs of similar author names.

SimilarAuthors

Author ID 1	**Name 1**	**Author ID 2**	**Name 2**	**Distance**
1	Kanamori	3	Kaanamori	1
1	Kanamori	5	Kanamori	0
1	Kanamori	7	Knamoqri	2
2	*Heiken*	*4*	*Reeken*	*2*
3	Kaanamori	5	Kanamori	1
5	Kanamori	7	Knamoqri	2
6	others	8	Yothers	1

(e) Groups of names that refer to the same author.

GroupedAuthors

Group ID	**Author ID**	**Name**
1	1	Kanamori
1	3	Kaanamori
1	5	Kanamori
1	7	Knamoqri
2	*6*	*others*
2	*8*	*Yothers*

2 Motivating Example

Consider that through the use of an extraction software, we have placed publication references in a database table, from which, we now want to obtain a list of unique author names. We start from an instance of this database table named PublicationAuthors as in Figure 1a that contains two attributes: (i) *Pub ID*, a publication identifier and (ii) *Authors*, containing the last names of each publication author separated by commas. The final expected result is the table CleanAuthors, shown in Figure 1b, whose schema has two attributes: (i) *Author ID*, an author identifier and (ii) *Name*, containing a unique author last name. In order to obtain such table, we have to execute a typical Approximate Duplicate Elimination process composed by the following high level data transformations:
Splitting: Splits each field value into a different row (another common option is splitting a field value into columns) using a character or pattern as a separator. In the example, Splitting extracts the individual author names from a string field using a comma as a separator (Figure 1a to Figure 1c).
Approximate Duplicate Detection: Computes a distance (or similarity) value using a distance (or similarity) function (e.g., the edit-distance [13] func-

tion used in the example) for all possible pairs of input rows (input *Name* field in the example). Then, it filters the rows using a given threshold value, e.g., distance values below 3 in the example (Figure 1c to Figure 1d).
Grouping: Computes a Transitive Closure algorithm in order to group similar records, e.g., in the example to assign similar names to the same group (Figure 1d to Figure 1e).
Consolidation: Finds a unique representation for each group using some criteria, e.g., selecting the most frequent name for each group (Figure 1e to Figure 1b).

An Approximate Duplicate Elimination process is typically modeled as a workflow of data transformations. The design of such a workflow is an iterative task where the designer first creates a process based on the data. Then, after verifying the results, he/she refines this process by adding, removing or modifying the data transformations, e.g., applying a more accurate criteria to describe approximate duplicate records.

Eventually, it is not sufficient to refine the data transformations to obtain more accurate cleaning criteria, then domain knowledge provided by the user is required during the process for properly cleaning data. So, it is important to let the user interact during the process by allowing him/her to manually clean part of the data generated by the automatic data transformations. For instance, in the example, filtering tuples of SimilarAuthors table in Figure 1d that do not correspond to real duplicates is not possible to be achieved by refining the criteria to discover the approximate pairs (e.g., changing the similarity function). In fact, the user must inspect all pairs whose distance is between 1 and 2 and decide the ones that do not concern the same real-world entity. After analyzing the pairs, the user deletes the non approximate duplicate pairs of author names, like the pair {*Heiken, Reeken*}.

Consider again the example. The GroupedAuthors table, Figure 1e, contains an invalid author name *others* (i.e., tuples that belong to group 2). To resolve this situation, first, the user needs to understand the provenance of the *others* value in the GroupedAuthors table. To perform this task, the user can rely on a debugging process, more concretely to go backwards in the workflow of transformations, and then understand that the *others* value in the GroupedAuthors table were generated due to the tuple in PublicationAuthors table, Figure 1a, with *Pub ID* value equal to 3. The user can conclude that, in this domain, the *others* value is often considered by extraction software as a valid author name. In order to prevent that an author name *others* appear in the CleanAuthors table, the designer could specify the program to automatically remove all *others* values produced in Authors table, Figure 1c, e.g, an instruction that eliminates *Names* values equal to *others*. Thus resulting in a new GroupedAuthors table without the tuples marked as italic shown in Figure 1e.

3 Tools for Approximate Duplicate Elimination

In this section, we describe the tools that we compare, Pentaho Data Integration in Section 3.1, CLEENEX in Section 3.2, and finally, OpenRefine in Section 3.3.

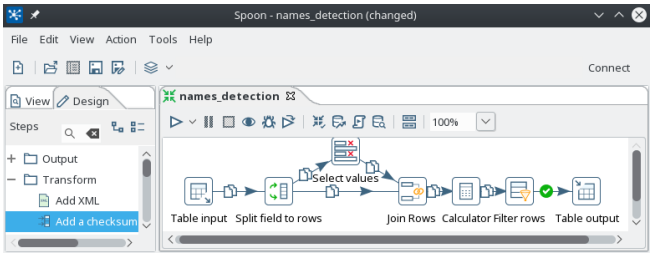


Fig. 2: Pentaho Data Integration Graphical User Interface

3.1 Pentaho Data Integration (PDI)

Pentaho Data Integration (PDI) is the ETL tool of the Pentaho Company. PDI offers more than 200 data operators, named steps, that allow the user to design and execute data cleaning processes, besides ETL processes.

PDI provides a Graphical User Interface (GUI) as represented in Figure 2. PDI models an ETL process as a data flow direct graph, whose nodes are steps which can extract, transform, filter and load data. The data flow between steps is modeled using direct links.

PDI can be used to design and perform data cleaning tasks, since it provides data transformation steps typically used in data cleaning programs. A data cleaning program in PDI that corresponds to the Splitting and Approximate Duplicate Detection transformations of the motivating example is shown in Figure 2. These transformations are combinations of the following steps: (i) a *Table input step* obtains the table *PublicationAuthors* from a database and injects each tuple into the data flow; (ii) a *Split field to rows step*, following the *Table Input Step*, extracts, into the data flow, the author names for each tuple of table *PublicationAuthors*; (iii) a *Select values step* is used to duplicate the data flow output of the *Split field to rows step* and to rename the attributes because the next step cannot receive exactly the same data flow twice with equal schema; (iv) a *Join rows step* computes a cartesian product between two data flows: one produced by the *Split field to rows step* and the other produced by the *Select values step*, thus generating each possible pair of names; (v) a *Calculator step* obtains the pairs of names produced by the *Join rows step*, and computes the edit-distance between those names; (vi) a *Filter rows step* outputs only the pairs of names from the *Calculator step* whose distance is below 3; and finally (vii) a *Table output step* is used to insert the data flow into a database table.

3.2 CLEENEX

CLEENEX is a data cleaning framework aimed at separating the logical specification of a data cleaning program from its physical implementation. It supports 5 data operators. CLEENEX provides a Graphical User Interface, illustrated in Figure 3, that enables the user to develop data cleaning programs. The development of a data cleaning program in CLEENEX involves the design of a data

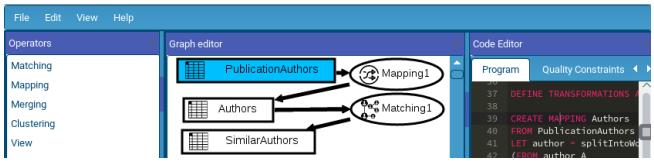
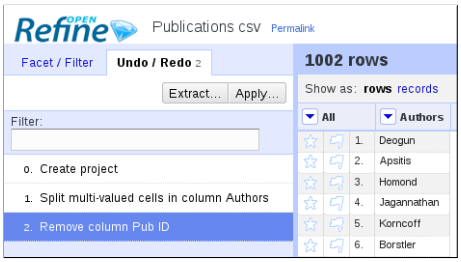
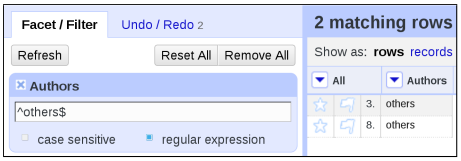


Fig. 3: CLEENEX Graphical User Interface



(a) Example of Splitting



(b) Example of applying a text filter

Fig. 4: OpenRefine Graphical User Interface

cleaning graph where the nodes are data operators or relational tables and the edges relate relational tables to data operators as input or output.

The designer can also specify Quality Constraints and Manual Data Repairs over any relation of a data cleaning graph [10]. Quality Constraints (QCs) assess the data quality in a specific point of the graph while Manual Data Repairs (MDRs) allow the user to apply a set of specific actions on the data tuples stored in any relation. The actions can be (i) updating attribute values, (ii) deleting tuples and/or (iii) adding tuples. CLEENEX also supports a debugging mechanism that enables the user to select tuples from any relation and navigate backwards through the graph displaying the provenance of these tuples.

Figure 3 contains a data cleaning graph in CLEENEX that corresponds to the Splitting and Approximate Duplicate Detection transformations of the motivating example. This graph is composed as follows: (i) The *PublicationAuthors* table is represented in the graph; (ii) a Mapping operator extracts the author names from table *PublicationAuthors* and creates the *Authors* relational table; (iii) given the *Authors* table as input, a Matching operator computes the distance between author names, and creates pairs of names whose distance is below 3, that are inserted in the *SimilarAuthors* table.

3.3 OpenRefine

OpenRefine is the former GoogleRefine tool for data transformation and cleaning, later released as an open source project. A project in OpenRefine displays a table as shown in Figure 4a. Users are allowed to edit any data cell of this table. OpenRefine supports more than 10 data operators, similarity to ETL tools, as it

allows for example a user to split column values into new columns or rows (e.g., the Splitting transformation used to create the Authors table in Figure 1c). In OpenRefine, the user applies operations to the table and the table content is immediately updated accordingly and visualized. Since data cleaning is an iterative process, if the user needs to refine the transformation being applied he/she uses a rollback mechanism (left windows of Figure 4a) that enables to undo operations.

OpenRefine also provides filters and facets to constrain the data rows. Filters and facets affect the visualization of the data tuples as shown in Figure 4b and also the data operations that are only applied to the constrained data.

4 Evaluation

In this section, we present the analysis and the experimental evaluation that we performed to compare Pentaho Data Integration (PDI), CLEENEX, and OpenRefine in terms of their support for the Approximate Duplicate Elimination problem. In Section 4.1, we present the evaluation criteria that we used, in Section 4.2, we list the experimental setup, and finally, in Section 4.3, we detail the analysis and the results of the performed comparisons.

4.1 Evaluation Criteria

In this section, we present the criteria used to evaluate the tools in what concerns their support for Approximate Duplicate Elimination. We grouped these criteria into three groups: (i) the program specification, (ii) the program execution, and (iii) the user utilities. Orthogonally, we further categorize the criteria used into Qualitative (QL) or Quantitative (QT) criteria.

The analysis of the program specification was performed through the following three criteria:

Extensibility (QL): We assessed the ability to extend a tool, e.g., the possibility to apply a new distance function to identify approximate duplicates.

Expressiveness (QL): We identified the capability of a tool to perform all the data transformations involved in an Approximate Duplicate Elimination task.

Conciseness (QT): We counted the minimum number of operators needed to specify the same Approximate Duplicate Elimination process for each tool.

The analysis of the program execution was performed through the following criterion:

Execution Time (QT): For each tool, we executed the Approximate Duplicate Elimination process and measured the execution time in seconds (CPU time).

The analysis of the user utilities was performed through the following two criteria:

User Involvement (QL): For each tool, we analyzed the mechanisms available to enable the user interaction during the execution of an Approximate Duplicate Elimination task.

Debugging (QL): For each tool, we analyzed the mechanisms available to allow a user to understand the provenance of a certain tuple generated during an Approximate Duplicate Elimination process.

Table 1: Support of extensions that can be required to specify an Approximate Duplicate Elimination process for each tool.

Extensible	PDI	CLEENEX	OpenRefine
Data Extraction Patterns	✓	✓	✓
Detection Functions	✓	✓	✗
Consolidation Functions	✓	✓	✗
Data Operators	✓	✗	✗

4.2 Experimental Setup

For evaluating PDI, CLEENEX, and OpenRefine using the quantitative criteria described in Section 4.1, we used the following experimental setup:
Dataset: We created a dataset named *PublicationAuthors* as presented in the motivating example that contains 113 publications and 10,000 author last names with an average of three author names per publication. The dataset contains 3,289 distinct authors and each author participates on average in 3 publications.
Error-generation: We used TDGen [3], an error-generator software application to introduce errors into the author's names of the dataset. The errors were placed randomly and with equal probability. The types of errors introduced are: Drop Last One or Two Characters, Typographical (insertion, deletion, substitution, and transposition of a character), Phonetic (applies a phonetic error pattern), and OCR (applies an OCR error pattern). The resulting data contains 3,324 modified author names with an average edit-distance of 1.5 to the original value.
Data Cleaning Program For each tool, we specified the same Approximate Duplicate Elimination process using the minimum number of operations as described in the motivating example.
Software: We used PDI community version 7.0, CLEENEX last development version, and OpenRefine version 2.6-rc2.

4.3 Analysis and Experimental Evaluation

Extensibility To overcome limitations in terms of the operations supported by a tool, in Approximate Duplicate Elimination, the designer may need to provide new data extraction patterns for Splitting, new detection functions for Approximate Duplicate Detection, new consolidation functions for Consolidation, or new data operators to provide an unsupported data transformation. In Table 1, for each tool, we report the support for those possible extensions.

In summary, by providing a programmable data operator in Java, PDI is the tool that is the most extensible followed by CLEENEX. OpenRefine can just be extended with new data extraction patterns.

Expressiveness Data cleaning program designers typically specify an Approximate Duplicate Elimination task using the operators supported by a specific tool. PDI does not supports the specification of the Grouping transformation without

Table 2: Number of operators/steps used to Approximate Duplicate Elimination the PublicationAuthors dataset using each tool.

High level transformations	**PDI**	**CLEENEX**	**OpenRefine**
Splitting	**1**	**1**	**1**
Approximate Duplicate Detection	4	**1**	5
Grouping	N/A	**1**	
Consolidation	5	**1**	
Full Approximate Duplicate Elimination task	N/A	**7**	9

an extension. CLEENEX and OpenRefine support all the high level transformations involved in Approximate Duplicate Elimination. Still, OpenRefine can not use detection criteria over more than one column.

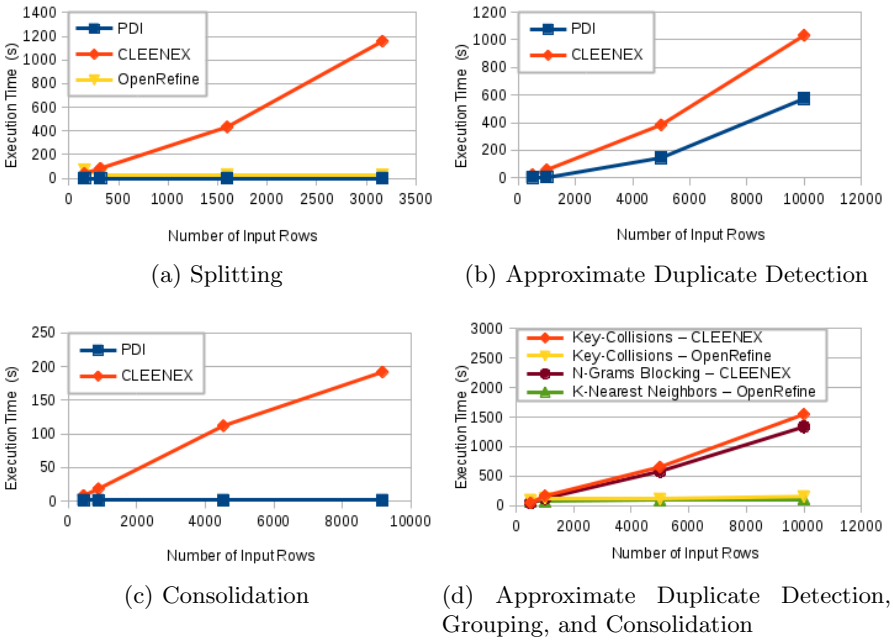
Conciseness For each tool, we evaluate the conciseness by reporting the minimum number of operators or steps required to specify each high level transformation and to specify the full Approximate Duplicate Elimination task as described in the motivating example. The results of this experiment are shown in Table 2, where in the first column we listed the high level data transformations and the full Approximate Duplicate Elimination task. In the remaining columns, we report the minimum number of operators or steps used in PDI, CLEENEX, and OpenRefine. For clarification, to specify a full Approximate Duplicate Elimination task that includes data import/exports are required additional operations. Also, N/A identifies the impossibility to specify a particular transformation.

CLEENEX required the lowest number of operators to specify each high level data transformation and the full Approximate Duplicate Elimination task because it provides an operator for each high level data transformation. PDI is the least concise tool since its operators are mainly aimed to ETL tasks.

Execution Time We report and analyze the results of the experiments performed to measure the execution time for each tool when executing the Approximate Duplicate Elimination process for the PublicationAuthors dataset.

Using PDI, CLEENEX, and OpenRefine, we report the execution times for Splitting in Figure 5a, Approximate Duplicate Detection using the Cartesian Product with the edit-distance in Figure 5b, and Consolidation in Figure 5c. In Figure 5d, we report the execution times of performing at once the three high level transformations: Approximate Duplicate Detection, Grouping, and Consolidation because in OpenRefine these transformations can only be executed at once by a set of operations. We performed the experiments using two different Approximate Duplicate Detection settings: (i) using the N-Grams Blocking [4] in CLEENEX and the K-Nearest Neighbors[9] in OpenRefine, both with the edit-distance function and (ii) using the Key-Collisions with the Fingerprint function[9]. Note that, the N-Grams Blocking followed by a Transitive Clustering in CLEENEX and the K-Nearest Neighbors in OpenRefine generate approximately the same results, since both rely on the comparison of n-grams.

[9] https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth



(a) Splitting

(b) Approximate Duplicate Detection

(c) Consolidation

(d) Approximate Duplicate Detection, Grouping, and Consolidation

Fig. 5: Execution times in seconds for performing each high level transformation.

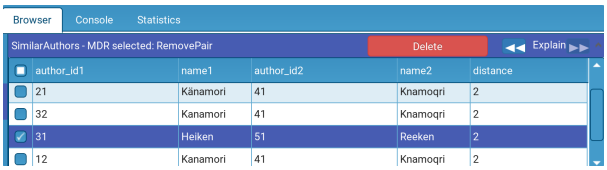


Fig. 6: Example of CLEENEX Manual Data Repair.

For the Splitting transformation, PDI showed to be the fastest tool followed by OpenRefine and the execution times from both tools did not grow with the number of input rows. In the remaining transformations, PDI and OpenRefine could not be compared since they did not share a common setting to execute the remaining transformations. CLEENEX is a single thread system with extensive database usage, not surprisingly, it was always the slowest tool.

User Involvement Enabling the user to interact during the execution of a data cleaning program to clean data can considerably increase the data quality. For instance, running the data cleaning program over the PublicationAuthors dataset without the user involvement generated 70,597 pairs of false duplicates.

In order to compare the user involvement in CLEENEX and OpenRefine, we considered the motivating example presented in Section 2 for removing the non

approximate duplicate pair *Heiken* and *Reeken*. Both CLEENEX and OpenRefine support manual repairs. Nevertheless, to filter Approximate Duplicate Detection task results using the distance (e.g., consider the tuples whose distance is below 3 and above 1), it is only possible to perform in CLEENEX. Because OpenRefine performs the Approximate Duplicate Detection task using clustering, the distance value is not provided. In CLEENEX, the designer can write an MDR, that as shown in Figure 6, constraints the visualizing tuples by the distance and that enables the user to delete tuples.

PDI does not allows the involvement of the user in a data process. CLEENEX is the tool that better supports the user involvement in an Approximate Duplicate Elimination process. OpenRefine allows the user to edit and delete tuples but lacks the addition of tuples and also the evaluation by the user of tuples pairs that fall into a distance/similarity interval. Moreover, in OpenRefine the process has to restart from the point where the user modifications were applied.

Debugging We analyze the data debugging mechanism supported by the three tools, i.e., where the user aims to understand the source of a particular tuple in a target table. Available components in PDI do not allow the user to visualize the whole set of tuples returned by a step and so it does not support debugging.

Consider again the motivation example presented in Section 2. To understand the problem and finding the source publications that generated the *others* value in Figure 1e, the user has to conduct a debugging process. In OpenRefine, the user needs to: (i) rollback to the data table before performing the Clustering operation, (ii) open the Clustering windows again, (iii) search and take note of the values that generate the *others* value, (iv) because the *Pub ID* column is eliminated after Splitting, the user has to rollback to the data table before that, (v) filter in the *Name* column for the values that generate the *others* value as shown in Figure 4b, (vi) take note of the values in the *Pub ID* column associated to the filtered values, (v) rollback to the initial table, PublicationAuthors, and finally (vi) filter for the values in the *Pub ID* column to obtain the publications involved in the generation of the *others* value. Instead in CLEENEX, the user just has to select the tuples of GroupedAuthors table containing the *others* value and perform backward data derivation until he/she reaches the initial table.

PDI does not support debugging. CLEENEX has a debugging component that allows the user to easily navigate through the cleaning graph and presents only the desired tuples. OpenRefine enables the user to visualize the tuples and provides filtering components that one can use in a debugging process but whenever it rollbacks to older data versions, the user has to search for the tuples.

5 Conclusions and Future Work

In this work, we compared PDI, CLEENEX, and OpenRefine for performing an Approximate Duplicate Elimination task. With this purpose, we created a dataset, introduced data quality errors, and specified the same Approximate Duplicate Elimination process for each tool. Then, we defined and evaluated

Benchmarking Named Entity Recognition Tools for Portuguese

André Pires, José Devezas, and Sérgio Nunes

INESC TEC and
Faculdade de Engenharia, Universidade do Porto,
R. Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
andre.r.o.pires@gmail.com,
{jld,sergio.nunes}@fe.up.pt

Abstract. There has been much work in the Information Extraction field, particularly in the Named Entity Recognition task. However, for the Portuguese language, the implementations still perform below the results for other languages, as shown by the HAREM conferences. The goal of this work is to assess the current performance of well established tools, namely Stanford CoreNLP, OpenNLP, spaCy and NLTK, against a Portuguese dataset, specifically the HAREM gold standard collection. These tools were used with an out-of-the-box approach, meaning without any tuning. The results show that the referred tools can match the results of Portuguese state-of-the-art tools, presented in the HAREM conferences, specifically, Stanford CoreNLP with an F-measure of 56.10%, OpenNLP with 53.63%, spaCy with 46.81% and NLTK with 30.97%. Furthermore, a hyperparameter study was performed, improving over the default configurations, for about 2% in each tool, and almost 35% in NLTK's `MaxEnt` classifier.

Keywords: natural language processing, named entity recognition, text mining

1 Introduction

Named Entity Recognition (NER) is a sub-field of Information Extraction. Its main purpose is to identify and classify entities from unstructured text. This extraction is the first part of a typical information extraction pipeline, enabling further information extraction methods, such as semantic analysis or relation extraction. There is a need for having systematic evaluation, so that all NER systems have the same standards when evaluating their performance.

There are multiple techniques proposed to rank NER systems based on their ability to annotate text correctly. These techniques were defined and used in conferences such as Message Understanding Conference (MUC) [1], Conference on Natural Language Learning (CoNLL) [2], Automatic Content Extraction (ACE) [3] and HAREM Avaliação de sistemas de Reconhecimento de Entidades Mencionadas (HAREM) [4]. These conferences not only differ in their evaluation

techniques, for example giving score to partial matches or only scoring exact matches, but also on what is considered an entity, so they use different entity class sets. This makes it hard to compare different tools which participated in different conferences.

In this paper, we will present a structured approach for the consistent assessment of the mentioned tools, in order to provide an out-of-the-box comparison of their performance. And afterwards, a hyperparameter study in order to improve the baseline performance.

2 Related work

Initial work in this field was based on pattern matching techniques, for instance Rau [5] extracted company names with rules like the detection of company suffixes, such as "Inc." or "Corp.". Afterwards, dictionary-based approaches began to appear, where the main extraction algorithm was based on matching the words in a text with a gazetteer. An example of this approach was done in Portuguese by Sarmento [6], using the gazetteer REPENTINO [7]. Although this kind of extraction techniques provided good precision, results in terms of recall and F-measure where low and, furthermore, these techniques are difficult to manage because rules and dictionaries need to be constantly updated. This lead to the development of machine learning algorithms, where the most used methods are probabilistic techniques, such as Hidden Markov Models, for example by Bikel et al. [8], Maximum Entropy Models, like the framework by Carvalho [9], and Conditional Random Fields, with a Portuguese example by Amaral and Vieira [10]. Throughout the years there was a shift in focus, where machine learning techniques started to gain more interest. This is due to the scalability of these techniques and the amount of work required, however data in specific domains remains scarce. Machine learning supervised approaches continue to be the most used techniques, but recently many semi-supervised approaches, involving bootstrapping [11], started to appear, as they require less training examples.

The HAREM [4] conference's main objective was to evaluate the current state of the art for NER in the Portuguese language. Apart from the HAREM conference, there have been some experiments in evaluating NER tools with the HAREM dataset, for instance by Amaral et al. [12] who compared NERP-CRF with other publicly available tools for the Portuguese language. Other example is by Rocha et al. [13], who compared PAMPO with other tools. Both of these comparisons used a subset of HAREM's categories. Another comparison using some of the tools of this experiment, for English, was done by Jiang [14], showing that Stanford CoreNLP had the best result, in concordance with our experiments.

2.1 Tool descriptions

The main purpose of this experiment was to compare some of the main tools capable of performing NER in multiple languages, with particular focus on the Portuguese language. The main criteria for choosing the tools were:

- The tool has to be completely free.
- The tool has to be language and domain independent.
- The tool has to allow training a custom model for NER, with custom entity classes.

With these criteria in mind, we chose four different, well-established tools: Stanford CoreNLP [15], OpenNLP [16], spaCy [17] and NLTK [18]. These tools are described in Table 1, which shows that all are freely available, use either Java or Python as the main language and, apart from already having multiple default language models, all allow training with other languages.

Table 1. Overview of the assessed tools.

Tool	License	Version	Language	NER classifiers	Default language models
Stanford CoreNLP	GNU General Public License	3.7.0 (2016)	Java	Conditional Random Fields	Arabic, Chinese, English, French, German, Spanish
OpenNLP	Apache License	1.7.2 (2017)	Java	Maximum Entropy	Dutch, English, Spanish
spaCy	MIT License	1.7.2 (2017)	Python	Thinc linear model	English, German
NLTK	Apache License	3.2.2 (2016)	Python	Naive Bayes, Decision Tree, Maximum Entropy	English

Neither of these tools have a Portuguese model for NER, so all of them require training with a Portuguese corpus, such as HAREM. Apart from spaCy, all tools use well known algorithms to perform NER, for instance CRF and Maximum Entropy. spaCy uses its own implementation of a structured average perceptron, called Thinc [19] linear model.

3 The HAREM gold standard collection

HAREM is an evaluation contest for NER in Portuguese. There were two main HAREM events, in 2005 [4,20] and 2008 [21]. Both provided gold standard collections and, for this experiment, we used only the second one.

3.1 Corpus description

HAREM's gold standard collection [22] is a collection of Portuguese texts from several genres, such as web pages and newspapers, in which named entities have been identified, semantically classified and morphologically tagged in context. This collection is composed mainly of news documents (about 35%) and didactic documents (about 23%). It defines three levels of entity annotations, namely categories, types and subtypes.

There were 10 categories identified, specifically Works of art (*Obra*), Event (*Acontecimento*), Organization (*Organizacao*), Misc (*Outro*), Person (*Pessoa*), Abstraction (*Abstracao*), Time (*Tempo*), Value (*Valor*), Local (*Local*) and Thing (*Coisa*). Apart from the categories, it has a total of 43 types and 21 subtypes.[1]

Table 2 shows the number of named entities in each category in the second HAREM gold standard collection. *Pessoa* is the most common category with 2,035 words, and *Outro* is the least common, with only 148 words. This gold standard collection has a total of 129 documents, divided into two Portuguese variants, for Portugal and Brazil, as shown in Table 3, being Portuguese from Portugal the main variant.

Table 2. Number of named entities in each category. Values from do Amaral et al. [12].

Categories	Number of entities	%
Pessoa	2,035	28%
Local	1,250	17%
Tempo	1,189	16%
Organizacao	960	13%
Obra	437	6%
Valor	352	5%
Coisa	304	4%
Acontecimento	302	4%
Abstracao	278	4%
Outro	148	2%
Total	7,255	100%

Table 3. Document distribution. Table from Mota et al. [23].

Portuguese variant	Number of documents	%
pt_PT	93	72.09%
pt_BR	36	27.91%
Total	129	100%

3.2 Corpus transformation

The gold standard collection is available in XML format, and contains information that was not used in this experiment. The collection cannot be directly used as input to the selected tools, so it had to be transformed for each tool.

Initial transformation This first transformation was applied for all tools.

First of all, the tag `OMITIDO` was ignored in the HAREM evaluation, so it was stripped from the gold standard collection, keeping the text without any annotation. Then, since HAREM provides alternatives to the annotation of entity tags, regarding the entity boundary, using the `ALT` tag — in other words, allows multiple annotations to the same span of text with different boundaries —, we stripped these tags, keeping only the alternative which had the highest amount of entity tags, and choosing the first alternative when there were no tags inside the `ALT` tag. This was required because the selected tools cannot handle

[1] See full hierarchy table at http://www.linguateca.pt/aval_conjunta/HAREM/-tabela.html - Accessed on: 2017-06-01.

alternative annotations. Apart from the alternatives using the `ALT` tag, there were alternatives in the annotation, regarding the category, type and subtype of an entity. In this case, we selected the first alternative for each case. Also, there were tags which had no categories, and were identified only as an entity. In this case, the tag was stripped.

Since the tools only allowed one level of entities, we flattened the levels, producing three different outputs: one with only the categories, one with only the types and another with only the subtypes. For scripting purposes, the types and subtypes were concatenated to keep the semantics and context inside the parent category (e.g. "LOCAL_HUMANO_PAIS").

Finally, apart from the category attribute, all other attributes were removed. For evaluation purposes, this dataset was divided into folds for repeated cross validation (see Section 5).

4 Main steps to run each tool

Each tool has a particular set of requirements in order to train a NER model and then perform NER. The steps for each tool are presented next, namely the required input format, the steps for converting HAREM into that format, how to train the NER model, how to perform NER, and finally how to convert it to the CoNLL evaluation format.

4.1 Stanford CoreNLP

CoreNLP requires a tokenized file, where each line contains a token and its entity class separated by a tab character. This entity class is the class of the entity for entity tokens, and an "O" class for other tokens. We used the Stanford CoreNLP tokenizer `edu.stanford.nlp.process.PTBTokenizer` to tokenize the text. The entity classes in XML were also tokenized in the process, so afterwards we "de-tokenized" them and we looped through the file adding the entity classes from the first match in an entity tag to a match in the closing tag. Every token outside this, was tagged "O". The classifier was trained using the command:

```
java -cp stanford-corenlp.jar edu.stanford.nlp.ie.crf.\
    CRFClassifier -prop <file.prop>
```

where `file.prop` sets the hyperparameters and features to use and the path for the training file and output model.

To classify text documents, we used the following command:

```
java -cp stanford-corenlp.jar edu.stanford.nlp.ie.crf.\
    CRFClassifier -loadClassifier <ner-model.ser.gz> --\
    testFile <file_test.txt>
```

where `file_test.txt` represents the input unannotated text to be classified. Then, after performing the Named Entity Recognition (NER), we added IOB tags to the output for it to be comparable in the evaluation stage.

4.2 OpenNLP

This tool requires a sentence per line as input, where entities are annotated with a starting tag (`<START:tag-name>`) and an end tag (`<END>`). First, we converted the HAREM XML entity tags to the OpenNLP format. Then, using NLTK's sentence segmentation module, the dataset was segmented by sentences. Since this segmentation was not perfect, we had to join faulty segmentations by checking if every open entity tag had a corresponding closing tag and vice versa, in each sentence. When they had no corresponding tag, we joined the current sentence with the previous sentence. Furthermore, we had to make sure that there was a space character before and after each tag, or else OpenNLP's interpreter would not work. To train the model, we had to run the command:

```
opennlp TokenNameFinderTrainer -model <model.bin> -lang <pt>\
    -data <training_data.txt> -encoding <UTF-8>
```

and to classify the text the command was:

```
opennlp TokenNameFinder <model.bin> < <corpus_test.txt> > \ <
   output file>
```

Finally, after performing NER, we converted the output from the OpenNLP format to the CoNLL format, adding IOB tags.

4.3 spaCy

spaCy requires that the input dataset is in the standoff format, that is to say, there have to be two files: one with the plain text, and another with the entity annotations, containing a tab separated entry with the beginning and ending positions of the entity along with its class. The text had to be separated in sentences. Since we had already converted HAREM to the OpenNLP format, we used it to transform it to standoff. The transformation was done using the following steps:

1. Search until `<START:tag>` tag
2. Save starting position
3. Delete matched tag
4. Search for `<END>` tag
5. Save end position
6. Delete matched tag
7. Save `standoff = (beginPos, endPos, tag)`
8. Repeat from step 1 until no matches occur
9. Output to a tab separated file

The resulting files were then used to train a NER model in spaCy. The training script was based on an example script in spaCy's repository, with the additional preprocessing task of converting to the standoff format. The main NER classifier was changed from `EntityRecognizer` to `BeamEntityRecognizer`. After the NER process, the result was converted to the CoNLL format with IOB tags for evaluation purposes.

4.4 NLTK

This toolkit not only requires that the input dataset is in the CoNLL format with IOB tags, but also that it has the associated POS tag. In other words, the input file must be a tab separated file, where each line has the token, the POS tag and the entity tag in IOB format. This aspect required three major steps, namely tokenizing and performing POS tagging, tokenizing while keeping the entity tags, and joining both files. The steps for this were the following:

1. Tokenize and POS tag
   - Remove all tags from the HAREM dataset, in order to tokenize the text
   - Tokenize the dataset using `nltk.tokenize.word_tokenizer`
   - Using the resulting tokenized text, perform POS tagging
     - Train POS model using *floresta* corpus from `nltk.corpus`
     - POS tag resulting file from tokenizer step
2. Tokenize while keeping the entities
   - Tokenize the dataset using `nltk.tokenize.word_tokenizer`
   - Join consecutive tokenized entity tags
   - Transform dataset, matching the entity tags using regular expressions, assigning tags to each token
     - For each token, after an entity tag, assign a *B-tag*
     - For each token after the first entity token (previous step), or after an Inside (I) token, assign an *I-tag*
     - Assign an *O* tag for other tokens
3. Join POS tagged file with entity tagged file
   - Iterate through both files simultaneously
   - For each line, set *token POS-tag IOB-entity-tag*

To train the NLTK's classifiers, we used NLTK trainer[2]. This allowed us to run the following command:

```
python train_chunker.py <path-to-training-file> [--fileids\ <
    fileids>] [--reader <reader>] [--classifier <classifier\
    >]
```

We had to choose a different document reader, `nltk.corpus.reader.conll-.ConllChunkCorpusReader`, since the training files were in the CoNLL format. For this reader, we had to specify the entity categories in the NLTK trainer *init* file. Running the command resulted in a serialized model saved in the pickle format. In order to classify text, the model had to be loaded, the dataset to be classified was required to contain POS tag annotations and then classified with `chunker.parse(tagged)`. The parser returned the result in a tree format, which was converted to the CoNLL format using `nltk.chunk.util.tree2conlltags(ner_result)`.

[2] `https://github.com/japerk/nltk-trainer`

5 Evaluation

The method we used to assess the performance of each tool is described next. Each tool has its own evaluation scheme, however they are not directly comparable. To make it comparable we used a common evaluation scheme, the one used in the CoNLL [2] conference, where credit was only given to exact-matches or, in other words, both entity tags and boundaries had to be correct for it to count as a correct match.

Evaluation was done using the *conlleval*[3] script, taken directly from the website for the conference's occurrence in 2000. The script requires a file in the CoNLL format, with both the output of the NER tool and the gold standard. To be more precise, it is a space separated file, where each line contains a token, the gold standard tag and the predicted tag. It accepts files with or without IOB (Inside, Outside, Beginning) tags, being that, for the latter, each identification is treated as a single token entity. For this experiment, we evaluated the results with IOB tags.

Since not all the tools outputted the results with IOB tags, we added them whenever they were missing. After each run, we had to merge the outputs with the gold standard. For this merge to be correct, we had to use each tool's tokenizer for the testing set, or else the merge would not be successful as the tokens would not match correctly to the gold standard set.

For robustness, we used repeated 10-fold cross validation, with 4 repeats, and calculated the average of the precision and recall, and the macro-average for the F-measure, from every run. In other words, in each repeat, we split the dataset into 10 equal sized folds (in terms of documents), with different training and testing sets each. Then we ran each tool for each fold and for each repeat, and also for each level (categories, types and subtypes). Resulting in a total of $4\ repeats \times 10\ folds \times 3\ levels \times 4\ tools = 480\ runs$.

Finally, after running all the folds and repeats for each tool, that is to say training the models, and performing NER on the testing set, we evaluated the outputs using the CoNLL script and then computed the average for each level. This resulted in an average for each tool, each level, and each entity tag. The results are presented in section 6.

6 Results

The results from the repeated 10-fold cross validation are presented next. Table 4 represents the comparison among the tools for the highest entity class level (categories). It shows the average of three metrics (precision, recall and F-measure) for a single level of entity annotation, namely categories. Stanford CoreNLP takes the lead with an F-measure of 56.10%, followed by OpenNLP with 53.63% and then by spaCy (46.81%) and NLTK (30.97%). The results for NLTK are the ones obtained using the Naive Bayes classifier. It was NLTK's

[3] See `http://www.cnts.ua.ac.be/conll2000/chunking/conlleval.txt` - version: 2004-01-26

best result but it was the lowest score among all tools, below the other tools performance.

Table 4. Results for categories only.

Tool	Precision	Recall	F-measure
Stanford CoreNLP	58.84%	53.60%	56.10%
OpenNLP	55.43%	51.94%	53.63%
SpaCy	51.21%	43.10%	46.81%
NLTK	30.58%	31.38%	30.97%

Table 5 shows the comparison between the tools but with the average F-measure for all the levels. The ranking was similar to the categories results. In other words, Stanford CoreNLP remains on top in the category level, followed by OpenNLP, then spaCy and finally NLTK, in all levels. Again, NLTK's results remain far below in comparison to the other tools. Naive Bayes remained the best scoring algorithm amongst NLTK's algorithms. It is important to note that Stanford CoreNLP is highly computationally demanding, so we were not able to run it with the default configuration for the remaining levels, i.e. types and subtypes.

Table 5. F-measure for all levels.

Tool	Categories	Types	Subtypes
Stanford CoreNLP	56.10%	-	-
OpenNLP	53.63%	48.53%	50.74%
SpaCy	46.81%	44.04%	37.86%
NLTK	30.97%	28.82%	21.91%

Since NLTK provides different classifiers to perform NER, we present in Table 6 the average results for each classifier, for the highest level (categories). The `NaiveBayes` and `DecisionTree` classifiers have similar results, although `NaiveBayes` has slightly better results. `Maxent` (Maximum Entropy), however, performed worse than the other classifiers, with almost zero recall and F-measure. One possible explanation for these results is the default number of iterations (ten) that the algorithm goes through, or even the features used for training. OpenNLP's classifier also uses Maximum Entropy and it performed better, but it was configured to use 100 iterations in the default configuration.

6.1 Hyperparameter study

In order to improve the results obtained in the default configuration, we performed a hyperparameter study, where we checked the effects of some individual hyperparameters for each tool. Since the tools require lots of time to train NER models, it was not possible to perform repeated 10-fold cross validation as before,

Table 6. Results for the category level in NLTK, for all classifiers.

Classifier	Precision	Recall	F-measure
NaiveBayes	30.58%	31.38%	30.97%
Maxent	18.19%	0.58%	1.13%
DecisionTree	21.84%	25.72%	23.62%

so we only perform repeated holdout, again with 4 repeats, and a split of 70% training and 30% testing. This lead to different values for default configurations, in comparison with the baseline study. Table 7 presents a summary of the best obtained results, together with the default results.

Table 7. Summary results for all tools, category level.

Tool	Default F-measure	Best Configurations	Best F-measure
OpenNLP	50.90%	cutoff=4	52.38%
		iterations = 170	51.52%
SpaCy	45.70%	iterations=110	46.60%
Stanford CoreNLP	54.14%	tolerance=1e-3	54.31%
NLTK DT	26.14%	entropy_cutoff=0.08	26.36%
		support_cutoff=16	26.18%
NLTK ME	1.11%	min_lldelta=0, iterations=100	35.24%

Apart from NLTK's Naive Bayes classifier, all tools allowed tuning some hyperparameters. As we can see, for all tools we managed to improve the default performance, in particular, for the NLTK's Maximum Entropy classifier, there was an improvement of almost 35% (F-measure). This leads us to conclude that the default configurations for this classifier are not good, at all. The best performing models are published in INESC TEC's CKAN research data repository [24]. These models were trained with the HAREM dataset, producing three different models, one per entity level, for each tool, and can be directly used for Portuguese NER.

7 Conclusions and future work

With this experiment, it became clear that it is possible to take well established tools and use them to perform NER in Portuguese corpora. HAREM proved to be a good resource to be used as training and test sets for the assessed tools. However, since it is not updated since 2010, and given that the Portuguese language suffered alterations, with the orthographic agreement, future work on this area would be to create an updated gold standard collection.

This experiment provided a comparison for the baseline configuration of each tool, and showed the best individual configuration. In order to further improve

the performance, one has to experiment with feature engineering for each algorithm implementation. While it is important to note that these results are not directly comparable to the ones in HAREM since different methods of evaluation were used, the results showed that these tools perform similar to the results achieved in the HAREM conferences, with the best F-measure of 56.10% by Stanford CoreNLP, with a difference of only 1% in the highest scoring participant in HAREM. Also, the hyperparameter study proved to be beneficial, as we managed to improve on the baseline results.

Acknowledgements This work is partially supported by FourEyes, a research line within the project "TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020", financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF). José Devezas is supported by research grant PD/BD/128160/2016, provided by the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT).

References

1. Grishman, R., Sundheim, B.: Message Understanding Conference-6. In: Proceedings of the 16th conference on Computational linguistics. Volume 1., Morristown, NJ, USA, Association for Computational Linguistics (1996) 466
2. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the CoNLL-2003 shared task. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003. Volume 4., Morristown, NJ, USA, Association for Computational Linguistics (2003) 142–147
3. Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S., Weischedel, R.: The Automatic Content Extraction (ACE) Program Tasks, Data, and Evaluation. In: 4th International Conference on Language Resources and Evaluation, Lisbon, Portugal (2004) 24–30
4. Cardoso, N.F.P.F.: Avaliação de Sistemas de Reconhecimento de Entidades Mencionadas. Master's thesis, University of Porto (2006)
5. Rau, L.F.: Extracting company names from text. In: [1991] Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application. Volume i., IEEE Comput. Soc. Press (1991) 29–32
6. Sarmento, L.: SIEMÊS - A named-entity recognizer for Portuguese relying on similarity rules. In Vieira, R., Quaresma, P., Nunes, M.d.G.V., Mamede, N.J., Oliveira, C., Dias, M.C., eds.: Computational Processing of the Portuguese Language: 7th International Workshop, PROPOR 2006, Itatiaia, Brazil, May 13-17, 2006. Proceedings. Volume 3960 LNAI. Springer Berlin Heidelberg (2006) 90–99
7. Sarmento, L., Pinto, A.S., Cabral, L.: REPENTINO A Wide-Scope Gazetteer for Entity Recognition in Portuguese. LNAI **3960** (2006) 31–40
8. Bikel, D.M., Miller, S., Schwartz, R., Weischedel, R.: Nymble: a High-Performance Learning Name-finder. In: 5th International Conference on Applied Natural Language Processing, Washington, DC, Association for Computational Linguistics (1997) 194–201

Anonymization of Clinical Data

José Miguel Melo[1], Gabriel David[2]

1 Faculdade de Engenharia, Universidade do Porto
ei12050@fe.up.pt,
2 INESC TEC, Faculdade de Engenharia, Universidade do Porto

Abstract. Clinical research and public health studies require extensive and detailed access to the vast amount of health records being collected in most health related services. By studying clinical data, researchers are able to produce statistics, identify relations and learn trends in the health sector. These studies and data analysis are very important as they can bring great benefits and knowledge in healthcare. However, maintaining individual privacy is crucial for ethical and legal reasons.

To balance the need for rigorous data and the requirement of privacy protection, a study with a prototype on data anonymization is done and some models and algorithms are discussed. This study allows to propose and develop a practical way to efficiently anonymize clinical data. With this solution, the user can quickly and easily anonymize a given MongoDB dataset,through the provision of a set of configurations. The anonymization is done resorting to well known models and algorithms to protect privacy, associated with specific clinical criteria, restrictions and hierarchies. At the end of the anonymization, an anonymized version of the subset is obtained that meets the selected privacy model, balancing enough privacy versus keeping research value.

To solve some limitations imposed by the implemented prototype, some optimized versions are also presented. These optimizations allowed to improve performance and handle bigger datasets. All versions are compared in terms of performance and characteristics is done to better understand and decide which is the best version to a specific case. Quantified metrics of information loss are provided.

Keywords: clinical data anonymization, privacy models, arx tool

1 Anonymize Data for Privacy

With the growth of technology and data collection, came the necessity of sharing data for research purposes. However, with this, came the concerns about individual privacy. A good compromise between data dissemination and privacy preservation is anonymization of the data set. In the Health sector, the ability to fine tune this trade-off and to quantify the process is essential to minimize information loss and at the same time keep the required level of privacy. This paper studies the development of an anonymization tool for clinical data. The first section presents a brief state of the art. The second section covers the design of the tool, from requirements to the algorithm used in the prototype. The third section compares the results obtained with several versions and configurations. The paper finishes with some optimizations and conclusions.

In this first section some important concepts and tools are covered. These concepts are useful to understand the solution proposed in this article.

1.1 Principles

Data anonymization is a process in which a database that contains information about real people is converted into a new database that contains as much as possible of the original information, but in which it is not possible to identify any specific person. [1] This process always implies loss of information and there are several methods to anonymize a dataset. The best method is chosen according to a quality metric, which is used to evaluate the quality of generated data. [2,3]

The anonymization is defined using a privacy model, which is a condition that a database must satisfy to be considered anonymized.

There are three types of disclosure, which privacy data publishing aims to protect against: (1) **attribute disclosure** - sensitive information is discovered -, (2) **identity disclosure** - individual is associated to a row in the anonymized database - and (3) **membership disclosure** - determine if an individual is member of the anonymized dataset. [4] In the context of anonymization, there are several types of attributes that must be well-defined in the dataset: (1) *sensitive attribute* - attribute that must not be linked to an individual, (2) *identifiers* - directly identify an individual and (3) *quasi-identifiers (QID)* - attributes that can identify an individual if external information is available and used. [4, 5]

1.2 Privacy Models

During the last years, with the growth of data and the concerns about individuals privacy, multiple privacy models have been proposed and developed to minimize individual's privacy risk. In this section, two of them will be covered: k-anonymity and ℓ-diversity.

***k*-anonymity** k-anonymity was introduced by Latanya Sweeney in 2002 to solve the problem of producing a release of data that contains useful data about individuals while those individuals cannot be identified. [6, 7] It is the most

common and the most used privacy model because the results are satisfactory for most problems and it is the most practical and the easiest to achieve in most scenarios. It says that, in order to achieve anonymity, each group of *quasi-identifiers* must appear at least k times in the dataset. [7]

As an example, and looking to Tables 1 and 2, in the initial one the quasi-identifier *Age* has different values for all rows. To achieve k-anonymity with k=2, each value must appear at least twice. To achieve that, some generalization is done and two groups were created - [30-40] and [20-30) - leading to Table 2.

ℓ-diversity ℓ-diversity was proposed by Machanavajjhala et al. in 2007. This principle tells that there must be a minimum of ℓ distinct values on each sensitive attribute for every group of QIDs. When this is achieved, a table is said ℓ-diverse. [8]

When compared with k-anonymity, we notice that this model is harder to achieve and implies more information loss.

As an example, and looking to the same tables as in k-anonymity, the sensitive attribute *Disease* assumes 3 possible values:*"Heart Disease"*, *"Cancer"* and *"HIV"*. To achieve l-diversity, each QID should have at least l values for the sensitive attribute. The 2-diversity table shows exacly that, being that: (1) [30-40] contains *"Heart Disease"*, *"Cancer"* and *"HIV"*; (2) [20-30) contains *"Heart Disease"* and *"Cancer"*. This satisfies ℓ-diversity model with $\ell = 2$.

Age	Sex	Disease
37	Male	Heart Disease
34	Female	Cancer
31	Male	HIV
24	Female	Cancer
26	Male	Heart Disease

Table 1. Patient's diseases table

Age	Sex	Disease
[30-40]	*	Heart Disease
[30-40]	*	Cancer
[30-40]	*	HIV
[20-30)	*	Cancer
[20-30)	*	Heart Disease

Table 2. 2-diversity patient's diseases table

1.3 Approaches to anonymity

In order to anonymize a database, there are three main approaches:

1. **Generalization** - consists on replacing values with a more general value. This approach requires an hierarchy of values where each value has a corresponding value in the next generalization level. With deeper hierarchies, there is less information loss as each level in the hierarchy has more precision than if there were less levels of generalization.
2. **Suppression** - consists in entirely removing a value from the dataset. This approach can be seen as a maximum generalization level in which the value is replaced with a *, which is a value that does not bring relevant information.
3. **Perturbation** - technique in which data in a record is changed to a non-real value. This approach is not recommended for research purposes as it generates data that does not correspond to reality.

1.4 Quality Metrics

Anonymization process implies losing data quality and information. It is important not to forget about the quality of generated data and try to find a balance between anonymization and information loss.

Quality metrics of anonymization are an important part of this process and can be expressed in the form of information loss and utility.

Precision (*Prec*) The Precision (*Prec*) metric was introduced by Sweeney [9] with the aim to evaluate the information loss. This information loss is evaluated taking into account the amount of modification of a table caused by anonymization. This metric consists in calculating the ratio between number of generalizations made and total of possible generalizations. [9, 10]

Normalized Certainty Penalty (NCP) This metric, which was proposed by Jian Xu et al. [11], is based on the approximation of generalized entries to the original ones, which allows to evaluate the information loss caused by anonymization process. As it is based on the approximation of entries to the initial values, it separates the definition into the two main types of data models: *numeric* and *categorical*. Each of these main types has a specific way to be evaluated.

1.5 Flash Algorithm

Flash algorithm was proposed by Kohlmayer et al.. and uses the concept of lattice. According to its authors, this algorithm goes over the lattice using a bottom-up breadth-first approach. [12]

Generalization Lattice Generalization hierarchies for the QIDs tuples can be represented as a lattice, in which each node represents a possible version of the database. The optimal solution for the anonymization problem is within the lattice, corresponding to one of the nodes.

This algorithm uses predictive tagging to reduce the number of nodes to be examined. This means that a node is tagged after being processed, avoiding unneeded processing.

Flash algorithm iterates through every node and finds the path from that node to the next node that only has tagged successors. If that condition is not fulfilled, the path will be from the node in question to the top node. The created path is checked using binary search so that anonymous nodes are tagged and non anonymous are added to the heap. After the path check is done, the algorithm continues to next iteration using the heap nodes.

The algorithm ends when top level is the starting node. Figure 1, which was retrieved from its original article, shows how the algorithm iterates through the lattice and allows to better understand the above said. [12]

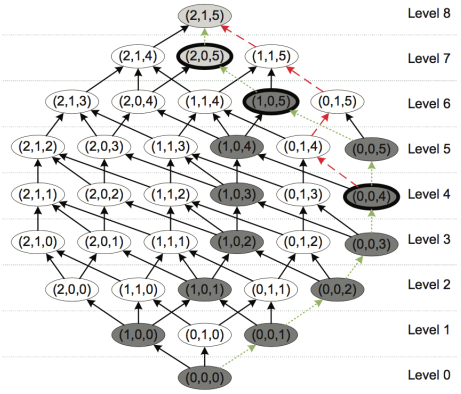


Fig. 1. Flash algorithm lattice example *(Source: [12]).*

1.6 Tools

Currently, there are already some tools to solve the problem of anonymization, such as:

- *PARAT* [13] - it is closed source and the available information to the public is limited.
- *Open Anonymizer* [14] - open source tool which uses k-anonymity.
- *UTD Anonymization Toolbox* [15] - toolbox with several anonymization methods implemented.
- *ARX* [16] - an open source graphical user interface (GUI) software for data anonymization.

1.7 ARX - Powerful Data Anonymization Tool

ARX is an open source software for data anonymization. With an intuitive cross-platform GUI, this tool can handle and anonymize large datasets. It also has a public API, which may be useful for other developers when implementing other software that require anonymization.

This tools has several features and supports: (1) Risk-based anonymization, (2) well known privacy models and quality metrics, (3) data transformation using generalization, suppression and microaggregation and (4) data utility analysis [17]

Although powerful, this tool only supports relational databases and does not contain predefined hierarchies for clinical data anonymization. For this reason, NoSQL databases, such as mongodb, can not be handled by this tool and in order to anonymize clinical data it is required a lot of configuration.

2 Solution

In this chapter, a tool for anonymizing clinical data will be presented, with the objective of providing the user with an easy to use interface that allows to quickly and effectively anonymize clinical data that is stored in a MongoDB database. This tool will take into account specific clinical data hierarchies and restrictions (e.g. *diseases* and *drugs*).

2.1 Requirements

In this section the requirements for the solution will be specified in more detail.

The solution must be applied to a Mongo database and should be scalable enough to be independent of the collection structure. All documents inside the MongoDB collection must have the same well-defined structure. The results must be stored into a new MongoDB collection with the same structure as the non-anonymized collection. The MongoDB database can be an intermediate step corresponding to the consolidation of several more traditional relational databases.

The anonymization process will run in a monthly basis as a way of clustering the process without losing too much information. However, in order to find out correlations between data, there must be a way to tell that two different records correspond to the same individual, but without compromising its privacy.

The system in which the process will run has limited amount of memory and processing units. So, the amount of data that can be stored into memory during the process is limited. The solution must be able to handle the maximum amount of data so it can handle the average of data collected each month.

In terms of speed and time of operation, it is essential to maintain a balance between the amount of resources required by the system and the processing time.

In order to handle different collections, the anonymization variables (*quasi-identifiers*, *identifiers*, ...) must be easily configured by the database administrator.

2.2 Architecture

From the requirements analysis, three main modules were defined for a better final solution:

1. **Anonymization process** - fetches a MongoDB collection, anonymizes it and stores the anonymized version in a new mongo collection.
2. **Anonymization GUI app** - a simple GUI that allows to easily create a configuration file and run the anonymization process.
3. **Anonymization web service** - allows the administrator to automate the anonymization and view some useful analytics about the process.

Anonymization Process The requirements analysis and the analysis of the state of the art allowed to better understand the problem and to define a better solution to solve the problem of clinical data anonymization.

ARX tool implements Flash algorithm and several privacy models - such as k-anonymity and ℓ-diversity. This tool provides a Java API to access these implemented privacy models and algorithm. This is useful for this solution as it allows to reduce the effort on implementing the already existing algorithm and privacy models.

Taking this into account, the anonymization process is divided into eight steps: (1) gather data; (2) process data - convert mongo hierarchical structure into a flat structure, making data compatible with ARX API; (3) initialize attributes; (4) anonymize data - call API provided by ARX, which implements Flash algorithm; (5) convert to the initial structure; (6) save anonymized dataset, (7) quality metrics calculation and (8) k-anonymity and ℓ-diversity calculation - way of validating the correctness of results returned from ARX.

These steps are represented as a flowchart in Fig. 2.

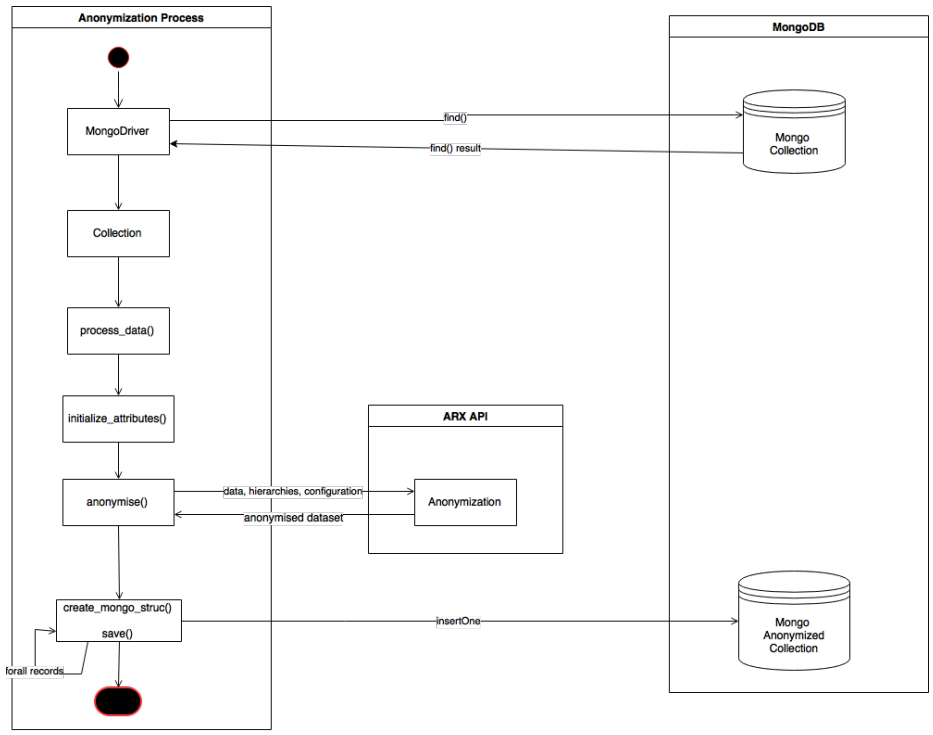


Fig. 2. Anonymization process flow chart

As input parameters, the anonymization process receives the MongoDB connection information (*Host*, *Database* and *Collection*), the destination collection information and the configuration file location. The configuration is a JSON file and contains all required information for the anonymization process - *quasi-identifiers*, *identifiers*, *sensitives*, *pseudonyms* and *excluded attributes* .

As output, it is returned a status message with information about the process, such as time taken to anonymize, number of records anonymized, final k-anonymity and ℓ-diversity and quality metrics.

Streams to reduce memory usage The anonymization process will run every month and the average of data collected per month is 400.000 records. This means that the solution must be capable of handling a large amount of data. It is important to search for solutions that can minimize the amount of Memory usage. Several possibilities were analysed, such as:

1. **clusters** - creation of smaller clusters that are anonymized independently and joined at the end. This possibility implies more information loss and splitting the monthly clustering with an even smaller cluster.
2. **unneeded attributes removal** - remove some attributes that are not required during the process of anonymization. Reduces the size of the dataset.
3. **indexing attributes names** - replace the name of each attribute with an index or id. Reduction in size is not significant.
4. **use of streams** - store data in memory only after all pre-processment has been done. Pre-processment is done using streams. This solution reduces memory but increases time of operation.

From the above possible optimizations and its analysis, it was chosen to join two of them: **use of streams** and **unneeded attributes removal**.

Clustering Pre-processing Phases Using streams minimizes the problem of handling bigger datasets but reduces the speed of the anonymization. Although speed of operation reduces, it does not compromise the well functioning of the solution while memory does.

In order to reduce the time to process data, parallel computing was considered during pre-processing stages - *gather data* and *process data*. During these phases, data does not need to be considered as a block to decrease information loss and clustering can be used. This is possible by using Sockets, which are used to transfer data between client and clusters. The basic functioning of this optimization is:

1. Client sends requests to clusters asking for pre-processment. Message includes mongo connection information, query to run, configuration used, page to anonymize and number of elements in that page. It waits for response from all clusters before continuing.

2. Cluster receives message from client, parses it and calls the data pre-processment method. Firstly, data pre-processment method fetches data from mongo collection. Then, it converts this collection, which contains an hierarchical structure, into a non-hierarchical representation of it. At the end, it returns to the client an array with the same structure as the one used by the API. In order to reduce the time lost sending data to the client, data is compressed before sending.
3. Client receives the array with all data already processed and joins it into the same structure. After all clusters answered, it continues the process of anonymization as in other versions.

In this solution, a thread is created for each cluster connection. In Fig. 3, it is shown a flow chart with the functioning of a client thread, of a cluster and the messages passed through sockets between those two components.

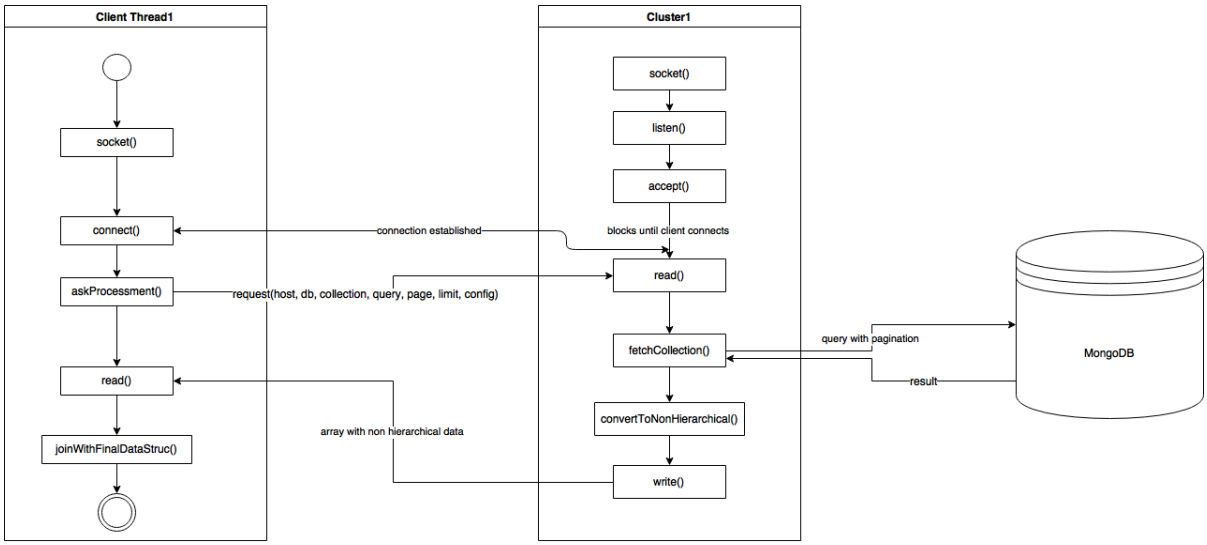


Fig. 3. Client-cluster functioning and interaction flowchart.

3 Results

In this chapter, the three versions are tested using distinct k and l values and several sizes for the dataset.

3.1 Information Loss

To test information loss, it is used two distinct configurations - k and ℓ - and the same dataset. This dataset has more than 70 fields, which means it is a big

dataset. These tests allow to compare k-anonymity and ℓ-diversity in terms of information loss.

***k*-anonymity** For this privacy model, three distinct k were used: k=2, k=3 and k=5.

# Records	$k = 2$	$k = 3$	$k = 5$
20k	Loss Information = 0.025 $Prec = 0.045$ Num. Suppressions = 237	Loss Information = 0.039 $Prec = 0.059$ Num. Suppressions = 521	Loss Information = 0.052 $Prec = 0.082$ Num. Suppressions = 1007
100k	Loss Information = 0.016 $Prec = 0.036$ Num. Suppressions = 234	Loss Information = 0.019 $Prec = 0.038$ Num. Suppressions = 500	Loss Information = 0.024 $Prec = 0.044$ Num. Suppressions = 990
600k	Loss Information = 0.014 $Prec = 0.034$ Num. Suppressions = 159	Loss Information = 0.014 $Prec = 0.034$ Num. Suppressions = 237	Loss Information = 0.015 $Prec = 0.035$ Num. Suppressions = 892

Table 3. Prescribed Medications anonymization - Information loss for 20k, 100k and 600k records using k=2, k=3 and k=5.

In Table 3, it is shown the resulting information loss for each k configuration and for each dataset size. From this table, it is possible to notice:

1. Information loss decreases with the increase of the dataset size - as example, when using $k = 2$, the information loss decreased from 2.5% for 20k records to 1.4% for 600k records.
2. Information loss is directly proportional to k.
3. Ratio of total suppressions has a significant decrease when using bigger datasets - as example, the ratio for fully suppressed records when using 20k records is 1.2%; when using 600k records it is 0.026%. (both for k=2)

For 600k records it is easily noticed that the information loss is almost the same for all k values. This means that the generated anonymized datasets are not too distinct from each other for all the three k versions. Charts on Fig. 4 allow to better notice this statement. Looking at them, we notice that, for 20k records (left chart), k has a significant impact in the anonymized dataset and the information loss highly depends on it. However, for 600k records that is not true and information loss is almost constant for every k. With this in mind, we can infer that for bigger datasets k=2 is preferable as the dataset is almost equivalent to the one generated using higher k and it has less suppression.

***ℓ*-diversity** For this privacy model, three distinct ℓ were used: ℓ=2, ℓ=3 and ℓ=5.

In Table 4 it is shown the resulting information loss for each ℓ configuration and for each dataset size. From this table and as expected, information loss (1) decreases with the increase of the dataset size and (2) increases with the increase of ℓ. Analysing in more detail each cell of the table, we notice that the evolution of information loss and amount of suppression is similar to the previous k-anonymity configuration. However, loss and suppression is a lot higher than for

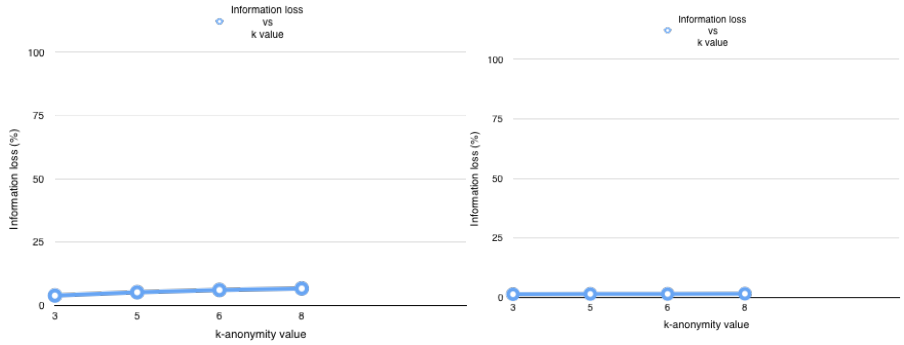


Fig. 4. Information loss vs k-anonymity for 20k (left) and 600k (right) records.

k-anonymity. These results confirm what was expected. ℓ-diversity ensures that each group of QIDs has a minimum of ℓ distinct values on sensitive attributes. So, the best scenario occurs when k anonymity also fulfills ℓ-diversity, which does not occur for this collection. As consequence, more generalization and more suppression is required to achieve ℓ-diversity and so more information loss.

# Records	ℓ **= 2**	ℓ **= 3**	ℓ **= 5**
20k	Loss Information = 0.584 $Prec$ = 0.809 Num. Suppressions = 6232	Loss Information = 0.751 $Prec$ = 0.900 Num. Suppressions = 8110	Loss Information = 0.841 $Prec$ = 0.928 Num. Suppressions = 11313
100k	Loss Information = 0.584 $Prec$ = 0.729 Num. Suppressions = 62573	Loss Information = 0.605 $Prec$ = 0.815 Num. Suppressions = 74388	Loss Information = 0.756 $Prec$ = 0.902 Num. Suppressions = 81378
600k	Loss Information = 0.465 $Prec$ = 0.533 Num. Suppressions = 211785	Loss Information = 0.525 $Prec$ = 0.708 Num. Suppressions = 337559	Loss Information = 0.547 $Prec$ = 0.789 Num. Suppressions = 425037

Table 4. *Prescribed Medications* anonymization - Information loss for 20k, 100k and 600k records using ℓ=2, ℓ=3 and ℓ=5.

3.2 Performance

In this section, the three versions are compared more directly in terms of speed and resources consumption. It was used an Intel Quad-core processor with 6Gb of RAM and a dataset with more than 60 fields.

Processing time Speed of operation is one important aspect to take into account. Table 5 presents the elapsed time for each version. On this table, one of the most notorious aspects is the *Out of Memory* in initial version for more than 100k records. However, if we focus on 20k records, the initial version is the best in terms of performance, with a duration of 10s. It is even faster than cluster-based solution (at least with up to 4 clusters). For this reason, initial version is the best choice when anonymizing small datasets and no clusters available.

For bigger datasets, and now comparing streams version with cluster-based version, we notice a big difference between these two versions. Cluster-based so-

# Records	Initial Version	Streams version	Cluster-based Version		
			2 Clusters	3 Clusters	4 Clusters
20k	10s	29s	16s	14s	12s
100k	Out of Memory	130s	56s	51s	49s
600k	Out of Memory	798s (13 min)	327s (approx. 5min20s)	308s (approx. 5 min)	296s (approx. 4min50s)

Table 5. Elapsed time comparison.

lution is notoriously faster than stream version. However, it has as disadvantage: the need of clusters to anonymize the dataset. For this reason, cluster-based solution is recommended for bigger datasets when there is the possibility to create clusters. Otherwise, stream version is the way to go.

Memory Usage Memory usage is one of the most important aspects to take into account as it is the most limiting factor for the anonymization process.

# Records	Initial Version	Streams version	Cluster-based Version
20k	1.7Gb	2.2Gb	1.2Gb
100k	Out of Memory	2.5Gb	1.7Gb
600k	Out of Memory	3.6Gb	2.4Gb

Table 6. Memory usage comparison.

Initial Version	Streams version	Cluster-based Version
84k	1.66M	3.1M

Table 7. Dataset size limitation comparison.

Table 6 presents the memory usage for each version and Table 7 the limitation in terms of dataset size. These tables show more clearly the evolution and the improvement done since initial version, passing from (1) out of memory for 100k records to wasting only 2.4Gb for 600k records and (2) maximum of 84 thousand records to 3.1 million records. This means an increase of 3590% on the dataset size that can be handled.

Concluding, from the analysis in terms of memory usage and processing time, we noticed a big improve in terms of performance since the initial version. Although this improvement, each version can be used in different situations: (1) although the first version requires more memory, it is the fastest and for this reason it can be used for smaller datasets (2) although both the second and third versions are slower, they require less memory to complete and for this reason it can be used for bigger datasets.

3.3 Validation

In order to validate the anonymization process results, two tests have been done:

1. **Test Description:** apply the anonymization process to an already anonymized dataset, using exactly the same configuration file.
   Expected Result: resulting dataset must be equal to the input dataset and, consequently, quality metrics must equal 0 (no difference between input and output).
   Results obtained: Both the input and output datasets are equal, which means the input dataset is already anonymized.
   Table 9 contains quality metrics calculated for each iteration. From this table, we conclude that when passing the already anonymized dataset, the metric calculation returns 0 (the input dataset did not suffer any alteration).
2. **Test Description:** calculate k-anonymity and ℓ-diversity for the anonymized dataset.
   Expected Result: k and ℓ must be equal or greater than the one passed in the configuration file.
   Results obtained: results for this validation test are presented in Table 8. From these results, we conclude that (1) for k configuration, the resulting k equals the input value and (2) for ℓ configuration, the resulting ℓ equals the input value and the resulting k is greater than ℓ. This allows to ensure the privacy model is guaranteed in the final dataset.
3. **Test Description:** ask some sample research questions to the anonymized dataset and compare the results with the non-anonymized dataset.
   Results obtained: the results were satisfatory and the questions could be answered using the anonymized dataset. For temporal variations, we noticed some information loss. However, this information loss was expected as dates were configured to monthly grouping.

These results allow to validate the output dataset complies with the requirements set by the admin and that the results can be used for research purposes which is the main goal of the usage of this solution.

Configuration	**# Records**	**k-anonymity**	**ℓ-diversity**
$k = 2$	100k	2	1
$k = 4$	100k	4	1
$\ell = 2$	100k	2515	2
$\ell = 4$	100k	4962	4

Table 8. k and ℓ returned from anonymization applied to Prescribed Medications dataset.

MongoDB collection	**# Records**	**Loss Metric**	**Precision Metric**
Non-anonymized	100k	2.4%	4.4%
Anonymized v1	100k	9.60E-6	9.78E-5
Anonymized v2	100k	0.0	0.0

Table 9. Quality metrics returned from anonymization applied to anonymized version of Prescribed Medications dataset.

4 Conclusions and Future Work

In the first part of this article, an analysis on the state of the art has been done and some concepts, algorithms, models and tools have been presented for protecting privacy while allowing data share for research purposes. This first part looked essential to understand how the problem of anonymization could be solved and allowed to propose a possible solution to correctly anonymize clinical data.

In the second part, a solution has been proposed based on the analysis done on the state of the art. This solution takes advantage of an API provided by ARX anonymization tool, which has the implementation of Flash algorithm. As input, the solution receives a mongo collection, which is processed to create a data structure and hierarchies that are compatible with the provided API. At the end of the process, the anonymized version of the dataset is exported to a new mongo collection. The solution has three versions: an initial version with all information being processed in runtime memory, another version using streams to process data, allowing to reduce the amount of memory used, and one last version using clusters associated with streams, which allows to reduce memory usage and improve speed of operation.

A web application has also been proposed to automate the process, which will occur once a month, and take control of it by providing the user with some useful analytics. These analytics allow the administrator to evaluate important aspects of the anonymization process, such as information loss, average of time taken and average of dataset size. A desktop application was also created intending to facilitate the creation of configuration files and to enable the user to run the anonymization process as a standalone application.

In the third part, the solution has been tested in more detail. From these tests it is possible to conclude that information loss increases for higher privacy model values and that it decreases with dataset size. Also, in terms of performance, the three versions of the solution have been tested and from these tests we concluded that each version has its own good and weak points. For this reason, each version can be chosen for a certain situation.

For the future, there is still some work that can be done to improve the solution created. First, in terms of performance, there is plenty of work to do to improve the consumption of memory during the process of anonymization. Second, it would be good to implement and use Big Data technologies to provide better results and to work on a scalable and distributed system. Third, it would be interesting to find ways of automate the process of learning new clinical dictionaries by implementing some machine learning algorithms.

Concluding, the research on the topic of clinical data anonymization allowed to better define the problem and find a good solution for it. By using a configuration file were all the process is configured, the solution fulfills all the requirements specified and is scalable enough to handle distinct datasets. Although the solution is already sustainable and fulfills the requirements initially specified, there is still some work that can be done to improve the solution.

Especificação de Interfaces Aplicacionais REST

Fábio Ferreira, Telmo Santos,
Francisco Martins, Antónia Lopes e Vasco T. Vasconcelos

Universidade de Lisboa, Faculdade de Ciências, LaSIGE

Resumo A programação de serviços *web* com interfaces aplicacionais REST é atualmente muito popular. O desenvolvimento de aplicações clientes destes serviços exige que as interfaces estejam bem documentadas. No entanto, e apesar de iniciativas importantes como a *Open API Specification*, o suporte à descrição destas interfaces é atualmente bastante limitado, focado essencialmente nos aspetos sintáticos. Neste artigo apresenta-se a linguagem HeadREST que permite ultrapassar estas limitações e descrever também os aspetos semânticos das interfaces num estilo reminescente dos triplos de Hoare e recorrendo a tipos refinados. A linguagem é apresentada através de pequenos exemplos extraídos de um dos estudos de caso que desenvolvemos com o intuito de a avaliar. Discute-se ainda a forma de validar a boa formação das especificações HeadREST e uma técnica para a geração de testes para verificar a conformidade de uma API REST relativamente à sua descrição.

1 Introdução

A programação de serviços *web* com interfaces aplicacionais que aderem ao estilo REST [5], designadas em inglês por *RESTful APIs*, e de aplicações consumidoras deste tipo de serviços é atualmente muito popular [8]. Por exemplo, aplicações como o Twitter, Instagram, Youtube e Uber, fornecem acesso programático às suas aplicações clientes através deste tipo de APIs. Isto acontece porque a utilização deste tipo de APIs, quando comparadas com as tradicionais interfaces de serviços *web* baseadas em SOAP, simplificam grandemente o desenvolvimento das aplicações clientes. Mais recentemente, com o advento da arquitetura baseada em micro-serviços, o desenho de aplicações como conjuntos de serviços tornou-se muito comum, alavancando ainda mais a utilização das APIs REST [3].

O desenvolvimento eficaz de aplicações clientes deste tipo de serviços exige que as suas interfaces estejam bem documentadas. Apesar de iniciativas importantes como a *Open API Specification*,[1] focada na criação e promoção de um formato aberto para a descrição de APIs REST, o suporte à descrição deste tipo de APIs é atualmente ainda bastante limitado e incide essencialmente na estrutura e representações dos dados trocados entre clientes e fornecedores.

De forma a ultrapassar as limitações existentes e suportar também a descrição dos aspetos semânticos subjacentes às APIs REST, desenvolveu-se a linguagem HeadREST que permite especificar cada um dos seus serviços individualmente, num estilo reminescente dos triplos de Hoare e utilizando tipos

[1] https://www.openapis.org

refinados. Trata-se de uma linguagem simples apenas com as características que foram identificadas como essenciais para ultrapassar as limitações das abordagens existentes. O objetivo da linguagem HeadREST não é assim estender a *Open API Specification* mas antes identificar primitivas que permitem alargar o seu poder expressivo e mostrar que é possível explorar estas descrições também para melhorar o estado da arte no que diz respeito à programação e teste de aplicações REST.

A linguagem é apresentada através de exemplos extraídos de um dos estudos de caso desenvolvidos para avaliar a linguagem: um sistema de gestão de labirintos, constituídos por quartos ligados entre si por portas. Este sistema, que possui uma interface aplicacional REST, encontra-se disponível em `https://mazes-demo.herokuapp.com/rest/v1/swagger.json`.

Discute-se ainda a verificação da boa formação das especificações HeadREST e uma técnica de geração de testes para aferir a conformidade de uma API REST relativamente à sua descrição. Os protótipos que implementam estas técnicas, ainda em desenvolvimento, tem a forma de um *plugin* para um *Interactive Development Environment* popular (o Eclipse) e utilizam uma ferramenta de verificação de teoremas (um SMT, o Z3 [2] em particular) para verificar as restrições geradas pelo sistema de tipos refinados e a relação de subtipos.

2 Contexto e trabalho relacionado

APIs REST O REST (*Representational State Transfer*) é um estilo arquitetural desenvolvido como um modelo abstrato da arquitectura da *web*, alicerçado no conceito de *recurso* [4]. De acordo com Fielding e Taylor [5], um *recurso* é uma função $M_R(t)$ que associa a cada instante de tempo t um conjunto de valores, os quais podem ser *identificadores* ou *representações* de recursos. Os identificadores servem para identificar o recurso envolvido numa interação. Para executar ações sobre um recurso, os componentes REST utilizam representações que capturam o estado corrente ou o estado pretendido do recurso.

No sistema que vamos usar como exemplo ao longo do artigo—um sistema de gestão de labirintos constituídos por quartos ligados entre si por portas—os recursos são labirintos, quartos e portas. Na figura 1 apresenta-se um exemplo do conjunto de valores associados a um quarto de um labirinto num determinado instante. Podemos ver que o quarto tem nesse instante dois identificadores (por um lado é o quarto 1 do labirinto 1 e, por outro, é o quarto de entrada no labirinto 1) e tem duas representações, uma em JSON e outra em XML.

Neste artigo focamo-nos em sistemas REST que comunicam sobre HTTP e que interagem com sistemas externos através de recursos *web* identificados por URIs (*Unique Resource Identifiers*). Assim, as ações que podem ser executadas sobre um recurso correspondem a pedidos de execução dos métodos oferecidos pelo HTTP—GET, POST, PUT e DELETE—e os meta-dados e dados a transferir pelo cliente são enviados, respetivamente, nos cabeçalhos (*header*) e no corpo (*body*) do pedido. Em resultado é produzida uma resposta com os dados e

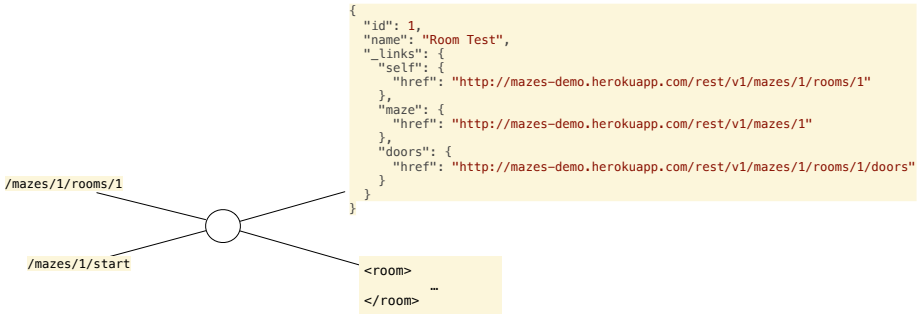


Figura 1. Um quarto de um labirinto num determinado instante

meta-dados a transferir para o cliente. A figura 2 mostra um exemplo de um pedido de execução de um PUT sobre o quarto referido anteriormente, identificado como o quarto 1 do labirinto 1. No pedido, para além de meta-dados, é comunicada ainda a representação (parcial) pretendida para esse quarto (ou seja, o novo nome). A resposta obtida, além de meta-dados, comunica a representação corrente do recurso sobre o qual foi realizada a ação.

As interfaces aplicacionais destes sistemas podem ser abstraídas como conjuntos de pares com um método e um *URI template*,[2] como ilustrado na primeira linha da figura 2.

```
PUT "http://mazes-demo.herokuapp.com/rest/v1/mazes/1/rooms/1"
-H "accept: application/hal+json"
-H "content-Type: application/json"
-d "{ \"name\": \"Room Test Updated\"}"
```

```
Code 200
Response body
{
  "id": 1,
  "name": "Room Test Updated",
  "_links": {
    "self": {
      "href": "http://mazes-demo.herokuapp.com/rest/v1/mazes/1/rooms/1"
    },
    "maze": {
      "href": "http://mazes-demo.herokuapp.com/rest/v1/mazes/1"
    },
    "doors": {
      "href": "http://mazes-demo.herokuapp.com/rest/v1/mazes/1/rooms/1/doors"
    }
  }
}
Response headers
content-type: application/hal+json
```

Figura 2. Pedido de execução de PUT sobre um quarto através do identificador `/mazes/1/rooms/1` e respetiva resposta

[2] `https://tools.ietf.org/html/rfc6570`

Descrição de interfaces aplicacionais As interfaces aplicacionais REST foram originalmente propostas por Fielding no contexto da abordagem *Hypermedia as the Engine of Application State* [4], a qual assenta na ideia de que o código dos clientes não é escrito de encontro a uma descrição estática de um interface do serviço, mas antes deverá utilizar um conjunto de pontos bem conhecidos de entrada no serviço para ir descobrindo dinamicamente quais as interações que o serviço permite. Porém, a popularidade do REST foi conseguida à custa de serviços que não aderem a esta visão e que fornecem antes interfaces estáticas contra as quais esperam que as aplicações clientes sejam programadas. Neste contexto é importante ter descrições das interfaces REST que possam ser usadas como documentação e, portanto, lidas por humanos e também processadas automaticamente, nomeadamente para gerar artefactos de *software* que facilitem o desenvolvimento das aplicações clientes ou dos próprios serviços.

Existem atualmente várias linguagens de descrição de interfaces (IDLs) que foram desenhadas propositadamente para suportar a descrição formal de APIs REST.[3] Entre estas destacam-se a *Open API Initiative*[4] (originalmente chamada *Swagger*), a *RESTful API Modeling Language*[5] (RAML) e a *API Blueprint*[6] pelo impacto e a quantidade de apoios que reúnem. Estas IDL permitem descrever com detalhe e rigor os aspetos sintáticos dos dados transferidos nas interacções REST e tem associadas uma grande panóplia de ferramentas, nomeadamente para a geração de documentação, geração de código de clientes em diferentes linguagens de programação e geração de testes. A expressividade destas linguagens é porém bastante limitada no que diz respeito aos aspetos semânticos. Por exemplo, recorrendo às especificações *Open API Specification* não é possível descrever várias propriedades da API que podem ser importantes. Recorrendo ao caso de estudo do sistema de gestão de labirintos, não é possível, por exemplo, expressar a propriedade que os nomes fornecidos na criação de labirintos e de quartos têm de ter entre 3 e 50 caracteres, nem que os nomes dos labirintos são únicos (i.e., que um labirinto só será criado se não existir outro com o mesmo nome), que os nomes dos quartos de um labirinto também são únicos e que não é possível apagar o quarto designado como a entrada de um labirinto.

Descrições focadas nos recursos são suportadas pelo RDF (*Resource Description Framework*), um modelo padrão para intercâmbio de dados na *web*, desenvolvido pela W3C.[7] Este permite descrever recursos e as relações entre estes (na forma de um grafo etiquetado), mas não permite descrever o comportamento dos serviços e da ação deste sobre os recursos. Uma perspetiva semelhante é a dos SERIN (*Semantic RESTFul Interfaces*) [6], uma IDL que permite definir modelos sintáticos e semânticos das APIs REST baseada na junção de anota-

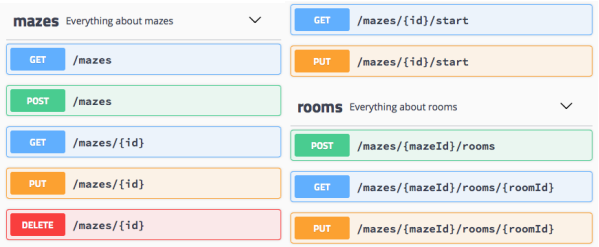


Figura 3. Excerto da API REST do sistema de gestão de labirintos

ções às classes das ontologias OWL *(Web Ontology Language)* para descrever que operações podem ser executadas e as restrições de integridade subjacentes.

O GraphQL é uma linguagem para consumir dados de APIs (não necessariamente REST), desenvolvida pela Facebook.[8] Permite aos clientes pedirem exatamente as propriedades de interesse, ao invés de se restringirem às representações disponibilizadas pelo servidor. Tal como no caso do RDF não permite descrever o comportamento dos vários serviços numa API.

3 HeadREST

A abordagem desenvolvida para especificar as interfaces REST assenta em duas ideias chave:

- a utilização de tipos para exprimir propriedades dos dados que são transmitidos nas diferentes interações e
- a utilização de pares de pré e pós-condições para exprimir (a) as relações que existem entre os dados enviados nos pedidos e obtidos nas respostas e (b) as mudanças de estado resultantes.

Estas ideias estão concretizadas na linguagem HeadREST, linguagem essa que é construída sobre dois conceitos fundamentais (cf. [1]):

- tipos de refinamento, `x:T where e`, que consistem em valores `x` do tipo `T` que satisfazem a propriedade `e` e
- um predicado, `e in T`, que devolve `true` ou `false` dependendo do fato do valor de `e` ser ou não do tipo `T`.

[3] Ver uma longa lista em `https://goo.gl/cY8knT`
[4] `https://www.openapis.org`
[5] `https://raml.org`
[6] `https://apiblueprint.org/`
[7] `https://www.w3.org/RDF/`
[8] `http://graphql.org`

Tipos de representação Para descrever a estrutura dos dados transferidos nas diferentes interações, consideram-se *tipos de representação*. Estes tipos, inspirados nos *modelos* usados nas especificações *Open API*, permitem ir além da estrutura dos dados e expressar propriedades sobre os seus valores.

Mais precisamente, os tipos de representação suportados são tipos de objetos (`{}` e `{l:T}`), tipos de vectores (`T[]`), tipos de refinamento (`x:T where e`), tipos escalares (incluindo `integer`, `boolean`, `string` e `URITemplate`) e o tipo de topo `any`. Estes tipos estão equipados com uma relação de subtipo definida semanticamente [1].

A linguagem de tipos base é extremamente rica, permitindo uma panóplia de tipos derivados. Por exemplo, o tipo singular, escrito `[e]`, é uma abreviatura do tipo refinado `(x:any where x==e)` quando `x` não ocorre livre em `e`. Uma utilização frequente deste padrão é `[null]`, o tipo que contém apenas o valor `null`. Tipos união `T||U`, intersecção `T&&U` e complemento `!T` são abreviaturas dos tipos `(x:any where x in T || x in U)`, `(x:any where x in T && x in U)` e `(x : any where !(x in U)`, respetivamente e onde `x` não ocorre livre nem em `T` nem em `U`. Utilizando os tipos intersecção podemos codificar tipos objecto multi-propriedade, como por exemplo o tipo `{l:T, m:U}` que aparece como abreviatura de `{l:T}&{m:U}`.

A linguagem de expressões conta com alguns operadores primitivos importantes. Entre eles encontra-se `matches` que verifica se uma *string* pertence à linguagem definida por uma dada expressão regular. Deste modo, o tipo `URI` dos identificadores de recursos é uma abreviatura de `(x:string where matches(e_uri, x))`, onde `e_uri` é uma *string* que denota a expressão regular descrita no RFC 6570.[9]

Por questões de conveniência de especificação, HeadREST permite associar nomes a tipos de representação e utilizar posteriormente esses nomes para referir os respetivos tipos.

```
type Link = {
    ?href: URI
}
type MRData = {
    name: (x: string where length(x) > 2 && length(x) <= 50)
}
type MazeGetData = {
    id: integer,
    name: string,
    _links: {
        self: Link,
        start: Link[] | [null]
    },
    _embedded: { orphanRooms: RoomGetData[] }
}
type RoomGetData = {
     id: integer,
     name: string,
```

[9] `https://tools.ietf.org/html/rfc6570`

```
    _links: {
        self: Link,
        maze: Link,
        doors: Link
    }
}
```

O exemplo introduz os nomes `Link`, `MRData` e `MazeGetData` e `RoomGetData`, os quais são associados a quatro tipos de representação: `Link` é um tipo de objeto apenas com uma propriedade opcional, com nome `href` e tipo `URI`; `MRData` é um tipo de objeto também apenas com uma propriedade cujo tipo é um refinamento de `string` que denota as sequências com um número de carateres entre 3 e 50; `MazeGetData` e `RoomGetData` são dois tipos de objeto com várias propriedades, todas obrigatórias, que utiliza o nome `Link` introduzido anteriormente (evitando assim ter de repetir a definição do tipo associado a esse nome).

Os tipos objecto podem ser estendidos em largura, juntando mais propriedades, através da relação subtipos, e.g., `{l:T} & {m:U} <: {l:T}`. Mas isto não restringe a `U` o tipo da nova propriedade `m`, sendo que por vezes, pretendemos fixar este tipo. É o caso do tipo `Link` onde a propriedade `href`, a existir, tem ser do tipo `URI`. O tipo `{?href: URI}`, definido como uma abreviatura de `(x:any where x in {href: any} => x in {href: URI})`, captura essa restrição.

Os valores de um tipo de representação podem ser apresentados de diferentes formas. Por exemplo, um valor do tipo `RoomGetData` tanto pode ser apresentado em JSON como em XML. No corpo da resposta apresentada na Figura 2, há um exemplo de um valor deste tipo apresentado em JSON.

Pedidos e respostas Como discutido anteriormente, nas interações dos sistemas REST há dados transmitidos no pedido (do cliente para o fornecedor) e na resposta (do fornecedor para o cliente). Seguindo de perto os princípios do REST, ao mesmo tempo que se abstraem detalhes considerados pouco relevantes, em HeadREST é considerado que qualquer pedido de execução de um método sobre um recurso é do tipo `Request` e que a resposta obtida é sempre do tipo `Response`. Estes tipo estão predefinidos na linguagem HeadREST e têm, respetivamente, a seguinte definição:

```
type Request = {
  location: URI,
  ?template: {},
  ?header: {}
}
```

```
type Response = {
  code: integer
  ?header: {}
}
```

HeadREST dispõe de duas variáveis predefinidas, `request` e `response` pertencentes aos tipos supramencionados.

Os dados transmitidos num pedido podem estar sujeitos a condições adicionais por o pedido acontecer sobre um recurso que é identificado por um URI, obtido por expansão de um determinado *URI template*. Essas restrições podem ser capturadas por um subtipo do tipo `Request`. Por exemplo, o tipo dos pedidos de execução de `GET` sobre uma expansão de `mazes/{mazeId}/rooms/{roomId}` inclui

sempre as propriedades `mazeId` e `roomId`, não havendo restrições para os valores que lhes estão associados. Isto é capturado pelo seguinte subtipo de `Request`:

```
{
  location: URI,
  template: {mazeId: any, roomId: any},
  ?header: {}
}
```

Estados Através de uma interface aplicacional REST um sistema expõe interações que permitem observar e modificar os recursos do sistema, ou partes destes. Assim, no contexto das especificações das APIs REST, considera-se que o conjunto de valores associados a cada recurso do sistema num determinado instante de tempo é o que define o estado do sistema nesse instante.

Assume-se que cada recurso é classificado com sendo de um determinado tipo, de entre um determinado conjunto de *tipos de recursos*. Por exemplo, no caso do sistema de gestão de labirintos precisamos de três tipos de recursos. Estes são declarados na especificação HeadREST da API da seguinte forma:

```
resource Maze, Room, Door
```

A figura 4 mostra um exemplo de um estado do sistema de labirintos. Neste estado apenas dois recursos não estão associados ao conjunto vazio de valores, um deles é um labirinto e o outro é o quarto de início desse labirinto. Este exemplo mostra ainda que os estados representam explicitamente o papel que cada valor associado a um recurso tem através das relações binárias primitivas `resourceidof`, `representationof` e `in`.

Asserções HeadREST permite descrever formalmente as observações e as mudanças de estado resultantes das interações expostas numa API REST através de um conjunto de asserções com a estrutura dos triplos de Hoare. Concretamente, as asserções são triplos da forma

$$\{\phi\}\ (a\ \ t)\ \{\psi\}$$

onde a é uma ação (`GET`, `POST`, `PUT` ou `DELETE`), t é um *URI template* e ϕ, ψ são expressões do tipo booleano. A fórmula ϕ, chamada de pré-condição, endereça o estado em que a ação é executada e os dados transmitidos no pedido enquanto que ψ, chamada de pós-condição, endereça o estado resultante da execução da ação e os valores transmitidos na resposta. A asserção diz: se o pedido para a execução da ação a sobre uma expansão de t transporta dados que satisfazem ϕ e a ação é realizada num estado que satisfaz ϕ então os dados transmitidos na resposta satisfazem ψ, assim como o estado resultante da execução da ação.

Por questões de espaço, apresenta-se a linguagem usada para escrever pré e pós-condições apenas através de alguns exemplos. Os exemplos escolhidos fazem parte da especificação da API REST do sistema de gestão de labirintos em linguagem HeadREST que foi desenvolvido como caso de estudo.

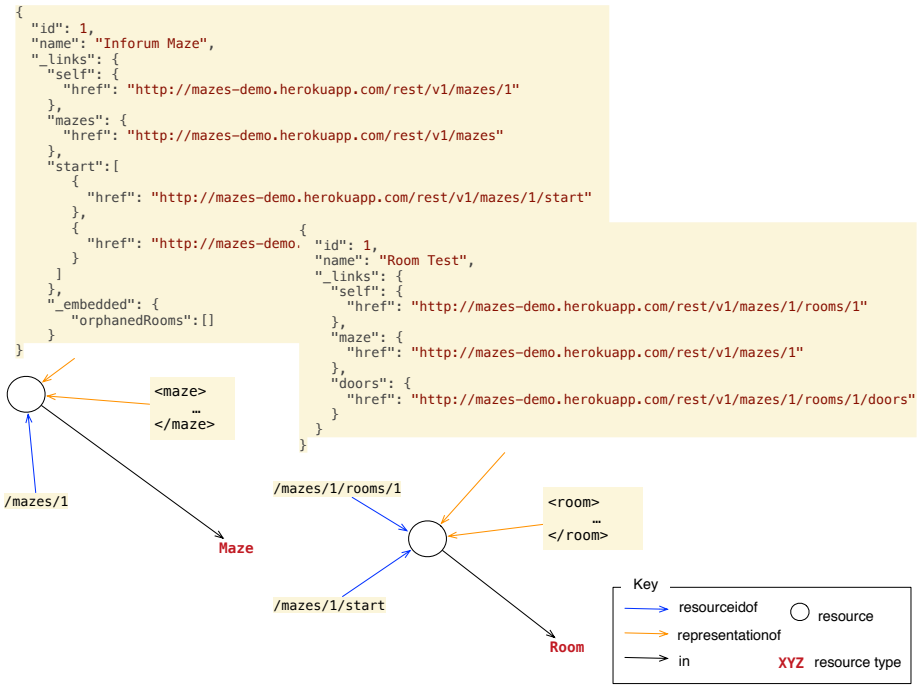


Figura 4. Um estado do sistema de labirintos

A primeira asserção endereça os pedidos de execução de `POST` sobre um recurso identificado por `/mazes` no caso em que os dados transmitidos no corpo do pedido são do tipo `MRData` (que como vimos anteriormente é um tipo objecto com uma propriedade com o nome `name`) e a ação é executada num estado em que ainda não existe qualquer outro recurso do tipo `Maze` com o nome indicado no pedido. A asserção declara que nesta situação o sistema efetivamente cria um recurso do tipo `Maze` com o nome indicado, que os dados transmitidos no corpo da resposta são uma representação desse recurso e o `URI` transmitido no cabeçalho um seu identificador, e que o código da resposta é o correspondente a `CREATED` (definido como 201). A asserção revela também que a representação do recurso acabado de criar que vem na resposta tem a propriedade `response.body._links.start` com o valor `null`.

```
1 // add maze, created
2 {
3   request in {body: MRData} &&&
4   (forall maze: Maze forall mazeRep:
5    (x: MazeGetData where x representationof maze) | mazeRep.
       name != request.body.name)
6 }
7   POST /mazes
8 {
```

```
9     response.code == CREATED &&
10    response in {body: MazeGetData, header: {Location: URI}}
          &&&
11    response.body.name == request.body.name &&
12    response.body._links.start == null &&
13    (exists maze: Maze | response.body representationof maze
14      && response.header.Location resourceidof maze)
15  }
```

A boa formação da asserção requer uma breve explicação. A boa formação de asserções é dada por um conjunto de regras que validam aspetos que não podem ser descritos sintaticamente. Em vez de apresentarmos as regras, apelamos à intuição do leitor. A linha 5 fala do objecto `request.body.name`. O tipo `Request` não nos permite concluir que a variável `request` tem uma propriedade `body` (e ainda menos que esta propriedade é um objecto com uma outra propriedade `name`). Mas o caso muda de figura se soubermos que `request` é do tipo `{body: MRData}`, porque `MRData` é um tipo objecto com uma propriedade `name`. A expressão `e &&& f` é uma abreviatura da expressão condicional `e ? f : false`. Deste modo a linha 5 está debaixo do ramo *then* do teste `request in {body: MRData}` (linha 3), o que nos permite assegurar que `request.body.name` está definido. Passando para a pós-condição, o operador `&&&` é usado crucialmente na linha 10 para estabelecer o tipo do objecto `response`, e permitir o acesso aos seus campos nas linhas seguintes. Além disso, o objecto `request` tem a propriedade `body` e esta a propriedade `name` (linha 11) porque o contexto da pré-condição estende-se à pós-condição.

A especificação inclui ainda duas outras asserções para o par `POST /mazes`: uma que endereça a situação em que já há um labirinto com o nome fornecido no pedido e uma outra para a situação em que `!(request in {body: MRData})`, ou seja o pedido não tem corpo ou tem mas os dados que vêm no corpo não têm o tipo certo. Em ambos os casos não é criado nenhum labirinto e os dados da resposta identificam o tipo de situação de erro ocorrida.

A asserção mostrada abaixo endereça por seu lado os pedidos de execução de `GET` sobre expansões de `/mazes{?page,limit}`. A asserção descreve que no caso de os valores dos parâmetros estarem dentro de determinados intervalos, a resposta ao pedido traz uma lista com as representações dos recursos existentes.

```
type RequestTemplate = {
  template: {
    page: (i: integer where 1<=i && i<=100000),
    limit: (i: integer where 1<=i && i<=50)
  }
}
// get mazes
{
    request in RequestTemplate
}
    GET /mazes{?page,limit}
{
    response.code == SUCCESS &&
    response in {body: MazeList} &&&
```

```
    response.body.meta.totalResults >= 0
}
```

4 Validação de especificações e geração de testes

A par da definição da linguagem HeadREST temos vindo a desenvolver uma ferramenta para verificar a boa formação das asserções e outra para testar APIs.

Validação de especificações A validação da boa formação das asserções baseia-se no trabalho de Bierman et al. [1]. A axiomatização original foi estendida com novas regras para definir a semântica subjacente às novas caraterísticas da linguagem HeadREST, nomeadamente recursos e vetores.

O verificador de tipos foi implementado como um sistema bidirecional que recorre a uma função para verificação e a outra para síntese [7]. No ponto de encontro entre a função de verificação e a de síntese encontra-se a relação de subtipos semântica, relação essa que é resolvida através de um SMT. A resposta do SMT é analisada, sendo emitida uma mensagem de erro ou de aviso. A geração de código de verificação a enviar ao SMT apresenta vários desafios, em particular, no que diz respeito à representação de *strings*, que endereçamos optando pela utilização da teoria implementada no Z3str2 [9]. Um outro desafio, sobre o qual ainda estamos a trabalhar, reside em conseguir uma axiomatização para o SMT que apresente uma eficiência razoável sem afetar a correção da mesma.

Geração de testes A técnica habitual de testes deste tipo de APIs envolve a criação manual de pedidos para cada operação, o que pode revelar-se um processo trabalhoso e conducente a erros, especialmente em APIs mais complexas. Ora, a partir da especificação da API é possível gerar casos de teste de forma automática que simplificam grandemente o processo de teste.

A nossa abordagem parte do princípio que o sistema a ser testado tem um estado inicial fixo e bem conhecido. Cabe ao testador construir um estado do sistema que satisfaça a pré-condição da asserção a testar. A partir daí, a nossa ferramenta (1) verifica se o estado a construir satisfaz de facto a pré-condição da asserção a testar, (2) gera testes para exercitar a API descrita por este axioma, e (3) executa os testes, construindo pedidos e verificando a pós-condição da asserção em teste.

Por exemplo, para a asserção que descreve a criação de labirintos apresentada anteriormente encontramos uma pré-condição que exige que não existam labirintos com o mesmo nome daquele que se pretende criar. A partir desta informação o testador tem de fornecer uma lista de operações que constrói este estado. Pode, por exemplo, utilizar a operação `POST /mazes` três vezes para construir um estado com três labirintos. A ferramenta coleciona as pré e pós-condições referente à criação dos três labirintos e com isso verifica a validade da pré-condição da asserção a testar. A geração dos dados para os pedidos é feita pedindo ao SMT valores dos tipos respetivos. Por último, a resposta da chamada ao serviço é verificada de encontro à pós-condição da asserção.

Fork-Work-Join: A Model for Executing Fork-Join Programs on a GPU

Alcides Fonseca[1] and Bruno Cabral[2]

[1] LASIGE, Faculdade de Ciências da Universidade de Lisboa, Portugal
amfonseca@fc.ul.pt
[2] CISUC, Departamento de Engenharia Informática da Universidade de Coimbra, Portugal
bcabral@dei.uc.pt

Abstract. GPUs have become a popular tool for parallel programming. When compared with multicore architectures, GPUs can offer higher throughputs. Typically, only data-parallel programs can be efficiently ported to GPUs. On the other hand, task-parallel programs often produce many divergent paths of execution, which raise concurrency problems resulting in an inefficient scheduling of GPU resources.
One of the most popular approaches to express task-parallel programs is the Fork-Join model, which fits perfectly on divide-and-conquer algorithms and typically makes an extensive use of recursive calls, which can be served simultaneously by multiple processors. Since generic GPU programming languages, such as CUDA and OpenCL do not support recursion, it is not easy to express fork-join programs on the GPU. Ideally, one should be able to express programs independently of the target platform and still make an efficient use of the available computing resources.
We introduce a new approach, which enables an efficient usage of the fork-join model in programs targeting both CPU and GPU architectures. The programming model combines three operations, which are supported by different backends in each platform: Fork, Join and Work. Fork and Join are lightweight operations that split calls or merge their results, while Work performs the heavyweight computations and enables recursion.
We have implemented two popular benchmarks, Fibonacci and Integral, showing that this approach can be used to express those types of programs. Those programs were executed on 2 different systems and the GPU version outperformed the multicore version of Fork-Join. We were able to achieve 2 times the speedup of the Fork-Join CPU version with 24 workers.

Keywords: GPGPU, Fork-Join, Irregular Programs

1 Introduction

As Graphics Processing Units (GPUs) evolved, they became quite popular in several domains different than graphical processing. With a large number of

cores and a specific architecture, GPUs provide a higher throughput than multicore processors at a lower price-point and with less energy consumption. Thus, developers have started to explore the usage of GPUs in computation intensive applications, leading to the field of General Purpose Programming on GPUs (GPGPU). Despite the higher throughput, GPGPU also has its disadvantages: there is a high latency in copying from the host memory to the GPU-specific memory; the memory layout is very different in the GPU, which may not fit all programs; GPU processing units cannot efficiently execute two instruction branches at the same time; and recursion is not commonly supported. NVIDIA CUDA 3.1 [1] introduced support for recursion, limited to certain GPUs, and through dynamic parallelism, which has a limited stack depth and is slower than regular calls.

Programmers have two options to write programs to use GPUs. Either they write a GPU-specific code optimized for their target platform, or they make use of a general purpose GPU framework. Writing GPU-specific code is very time-consuming and requires expertise in the area. For those without that skill, the solution is to use frameworks that provide a user-friendly API for common tasks, which are previously optimized, writing only the parts of the program specific for their domain. This skeleton-based approach is used by Map-Reduce frameworks such as Mars [2], Accelerate [3], ÆminiumGPU [4] or Matlab [5]. These frameworks use the Map-Reduce paradigm, which expresses high data-parallel tasks that can be efficiently executed on the GPU.

The Fork-Join model is also very popular in multicore frameworks, such as Cilk [6] and Java's Fork-Join [7]. Fork-Join fits the nature of divide-and-conquer algorithms that can be applied to a variety of fields, from sorting to numerical computing. However, the Fork-Join model was initially designed for CPU programming and, to the best of our knowledge, it has not yet addressed efficiently the GPU programming. For example, in recent work [8], Peloquin et al., showed a framework for GPU programming based in this model that had lower speedups when compared with the single-threaded version for CPU programming.

In this work we address this lacuna in general models for GPGPU, which prevents a whole set of programs from being written for the GPU, without expertise knowledge. Our research hypothesis is whether Fork-Join programs can be written and efficiently executed on the GPU.

This work contributes in two areas: First, we present the Fork-Work-Join model for expressing Fork-Join programs within the limitations of GPU programming languages. This model separates the actual work performed in each step from the forking or joining of results. Secondly, we present an execution model to efficiently execute those programs on the GPU. This is done using a pre-compilation phase to generate the GPU version of the application and by using a load-balancing runtime to handle irregular recursion.

In the remaining of this paper we introduce the Fork-Join and Fork-Work-Join models for the GPU (Section 2), how to execute those models on the GPU (Section 3). We briefly discuss the implementation of the proposed approach

(Section 4) and we evaluate it (Section 5). Finally, we compare this approach with related work (Section 6) and we draw conclusions (Section 7).

2 The Fork-Work-Join model

The Fork-Join [6] model has successfully been used for expressing parallelism in divide-and-conquer algorithms. It is usually defined using recursive functions, which may spawn new tasks for each recursive call. Tasks are joined before their result is used, thus guaranteeing that the result is available for the current iteration. In this section, we present the novel Fork-Work-Join model, an extension of Fork-Join for GPU and CPU execution.

2.1 Fork-Work-Join

It is advantageous to use the GPU only when massive parallelism is present, when the overheads of scheduling to the GPU are low compared to the computation. In order to adapt the Fork-Join model to the GPU, the problem should be split in an high number of subparts. The process of forking new tasks should happen as earlier as possible, in order to use the GPU high parallel throughput. Removing the heavy-weight work from forking allows to split the input several times before starting the larger part of the computations. A Fork-Work-Join program should define the three constructs:

- Fork/Splitting: allows for spawning tasks without performing any work.
- Work: performs any necessary computations and marks the task as being a base case. In that case no more recursive calls will be made, requiring no more splitting.
- Join/Combine: results from base cases are merged together.

```
public PList<Integer> split(Integer n) {
  PList<Integer> l = new IntList();
  if (n < 2) l.add(n); else { l.add(n-1); l.add(n-2); }
  return l;
}

public Integer call(Integer n, RecursiveCallback result) {
  if (n < 2) { result.markDone(); return 1; }
  return 0;
}

public Integer combine(Integer input, Integer other) {
  return input+other;
}
```

Listing 1.1: Fibonacci function defined using the Fork-Work-Join model

4

Listing 1.1 shows an example of the naïve algorithm for computing the Fibonacci sequence, using the Fork-Work-Join model. The `call` method checks if it is the base case and marks that subtree as completed. The spawning of new tasks is done in the `split` method. Finally, the `combine` method joins the result of the two calls together. This example is used to show how the division between Forking and Working is done.

3 Execution of Fork-Join models on the GPU

We present two approaches for executing Fork-Work-Join programs on the GPU. The first is a general model can be used in any recursive program. The second is a special case for programs that subdivide a range of numbers, which are used in several fields, such as finance, biology, physics and machine learning.

Recursive operations can be written following the Fork-Work-Join model. These operations are executed by the CPU following Algorithm 1. The algorithm initially tries to generate a large number of tasks in order to compensate the memory overheads required to use the GPU with enough work to gain speedup.

After the initial split, the CPU delegates the execution to the GPU. This delegation is wrapped in a for loop, in order to limit how much tasks are sent to the GPU. This limit is required due to the memory limits of GPUs. In each iteration of the loop, N elements are copied to the GPU where N is the number of GPU threads. The maximum size of the arrays is M, where $M = N * S$ and S is the maximum spawn level performed bu the GPU. S corresponds to how many spawns are in the number of iterations performed inside the GPU.

Inside each outter loop, memory is copied to the GPU and it executes two loops, each one in parallel. The first loop performs the *Work* phase and, if unable to complete, it spawns more work that overrides the current position in the arrays with its new first value and all remaining values are placed in the end of the arrays. The second GPU loop accumulates the completed results in parallel, using the reduce algorithm in AeminiumGPU [9].

A special and popular case of recursive algorithms are those where the subdivision occurs over a range of numbers. This range may be multi-dimensional. The range can also consist of integers that represent the positions of arrays, or of floating points for usage in numerical computations.

```
[CPU]
while length of tasks < number of threads do
    t = tasks.pop();
    tasks.append(op.split(args...));
end
while nt > 0 do
    copy tasks.arguments.pop(min(nt, N)) to GPU;
    [GPU]
    while 1 < nt < M do
        for i in 0..nt do
            accs[i] = op.work(arguments[i]);
            if controls[i] != COMPLETED then
                na = op.split(arguments[i]);
                for j in 0..length of na do
                    arguments[i + j * N] = na[j];
                end
                accs[0] = op.join(accs[0], accs[i]) ;
            end
        end
        k = 0;
        for i in 0..nt do
            if controls[i] == COMPLETE then
                accumulator = op.join(accs[0], accs[i]) ;
            else
                arguments[k++] = arguments[i];
            end
            controls[i] = INCOMPLETE;
        end
        nt = k;
    end
    copy arguments, accumulator to CPU;
end
return accumulator;
```

Algorithm 1: General Recursive Call Algorithm for GPU execution. **op** is the recursive operation; **arguments** is an array of arguments; **accs** is an array of intermediate results; **controls** is an array that marks if the operation is completed or not for each position; **nt** is the number of arguments that need processing

```
public Double getStart() { return 0.0; }
public Double getEnd() { return 1.0; }

public Double iterative(Double r, Double l, RecursiveCallback
    result) {
  double h = (r - l) * 0.5; double c = l + h;
  double fr = (r * r + 1.0) * r; double fl = (l * l + 1.0) * l;
  double fc = (c * c + 1.0) * c; double hh = h * 0.5;
  double al = (fl + fc) * hh; double ar = (fr + fc) * hh;
```

```
  double alr = al + ar; double prev = (fl + fr) * hh;
  if (Math.abs(alr - prev) <= RESOLUTION) { result.markDone(); }
  return alr;
}

@Override public Double combine(Double input, Double other) {
    return input + other; }
```

Listing 1.2: Integral function defined using the Fork-Work-Join model

A more concrete example is the Integral benchmark, using the trapezoidal rule to calculate the approximation of a function on a given range, until a certain resolution. This example divides the range of the integral in parts and recursively calculates the approximation of the new subrange. If the resolution is sufficiently good, recursion stops. The program is written using the Fork-Join Model in Listing 1.2. The advantage of this family of algorithms is that we can use the knowledge of the continuum nature of the arguments to avoid having an unlimited list of pending work. Instead, each GPU worker has a range of input that can be updated as soon as a subset is completed, by updating its lower bound.

For these cases, the arguments array contains the lower alimit of each section of the continuous range being calculated. Sections that are completed are removed from the arguments array, leaving space for uncompleted sections to be subdivided in new sections. This approach scales for multiple dimensions.

If at some point no section completes, a step variable is reduced by half, increasing the granularity of the algorithm, but leaving half of the work to be done, backed on an auxiliary GPU-only array. Once the first half of each section is done, the backed array will become the real array and the operation will resume, completing the second halves of each original section. To mark that a section has only performed half of its work, a *PARTIAL* result is used along with *Complete* and *Incomplete*. This optimization is available only for ranged recursive operations.

4 Implementation Details

We implemented the Fork-Join-Work model on top of an existing Map-Reduce framework for hybrid CPU and GPU. We used ÆminiumGPU [9], a framework for the JVM that translates lambda objects to OpenCL and executes programs in the GPU, in the CPU or in both. The decision is made automatically using machine-learning techniques applied to features retrieved at compile and runtime [4].

Figure 1 shows the pipeline of a program in the ÆminiumGPU framework. The first component is the special compiler that generates OpenCL

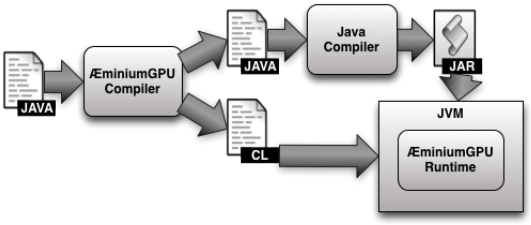


Fig. 1: Architecture of the ÆminiumGPU framework.

code for each special GPU-ready operations. The generated Java code is compiled to bytecode. During run-time, the runtime library handles the GPU initialization, memory copies and kernel scheduling.

In order to support the Fork-Work-Join model, we had to extend the compiler to also translate *strategy* objects in addition to Map and Reduce operations, following the same approach for Java to OpenCL translation. These *strategy* objects contain the three phases of the model Fork(*split*), Work(*iterate*) and Join(*combine*). These objects can be of two types: `RecursiveCall` and `RangedRecursiveCall`, with the first being the general case, and the second the specific case of ranged operations, as discussed previously. Each of the two operations has three possible backends: single-threaded (not in use), multi-threaded work-stealing Æminium Runtime [10], and a GPU version.

When evaluating recursive operations, the runtime system can split the work between the GPU and the CPU. It first splits the work in W working units, which are divided between the multi-threaded backend and the GPU backend. The value W defaults to $2 * N$, being N the maximum number of workers of each platform, so that all platforms can have work to occupy all workers twice. However this may not be possible if there are not enough recursive calls, in which W takes the maximum value possible. Each backend then executes independently and in parallel, until both are completed, the partial results are combined into the final output value.

The multi-threaded backend follows a standard binary Fork-Join model. In order to dynamically stop parallel recursion and switch to sequential recursion, a dynamic cut-off criterium is used. A further study on the impact of different criteria is detailed in [11].

The GPU backend uses a static cut-off threshold S, representing how many spawns can exist in a single worker. The specific case of ranged recursion has another static cut-off threshold D that represents how many iterations are performed on the GPU, before trying to load-balance tasks on the CPU.

In terms of arrays, only the argument array is copied to and from the GPU. The accumulator array is only copied at the end of the computation. The result array is copied to the host after every kernel call, in order to have information to perform the load-balancing. Instead of *COMPLETED* or *UNCOMPLETED*, the result control array stores whether that position is *FREE* or *READY*. If it is free, it means that we can use it for spawning a new task at that position. If it is ready, it means that it stores arguments to be processed. Instead of *COMPLETED*, the result is added to the accumulator, and the position is set as *FREE*. Since all of these verifications happen on the GPU, they do not require a global state, avoiding this way unnecessary synchronization bottlenecks.

5 Evaluation

In this section we describe the experiments performed to validate the performance of this approach, as well how several variables impact the performance of the applications.

5.1 Applications

To evaluate this model, we used programs from existing benchmarks that were irregular. Irregular programs are those that benefit from the Fork-Join model since the Map-Reduce approach is inefficient. Map-Reduce operations are only efficient when work is evenly balanced, and cannot express recursive subdivision of work upon a given condition. Fork-Join, on the other hand, is suitable for these kind of problems and provides a model for load-balancing of work.

To represent the general case, we selected the naïve Fibonacci program, widely used in the literature. This program has a light computing load, which makes it very useful to evaluate the cost of spawning. For the ranged operation, we used the Integral example, which uses the trapezoidal rule to approximate the integral of the function $f(x) = (x + 1) * x$ to a certain resolution. Both examples were also used to evaluate the Fork-Join framework [7].

5.2 Experimental Setup

In these experiments we used a machine with: 2x Intel(R) Xeon(R) CPU X5660 at 2.80GHz with a total of 12 cores and 24 hyper-threads; 24GB of RAM memory; NVIDIA Quadro 400, featuring 256 CUDA cores and 2GB of memory, with OpenCL 1.1; OS Ubuntu Linux Ubuntu 14.04.1 LTS and OpenJDK 1.8. Each execution was repeated 30 times and the JVM warm-up phase was excluded from all results.

5.3 Results

In Figure 2 we compare a Breadth-First (BFS) to a Depth-First (DFS) approach using FIFO and LIFO queues, respectively, on the GPU backend. The main advantage of DFS/LIFO on recursive programs is an early completion of subtrees, while FIFO can expand trees more easily. DFs has better scalability because it limits the number of executing tasks, reducing the overhead with auxiliary data for each computation created.

Another aspect of the implementation is the choice of two parameters: the number of workers on the GPU, and the maximum depth inside each worker. High values of these parameters result in an increase in memory usage, which limits the ability to speed the operations. And when these values are too low, it means that we are not taking advantage of all the resources. Figure 3 shows the effect of both parameters on the computation of Fibonacci of 40, from the two perspectives: Spawns and Workers. The optimal number of workers is 8192

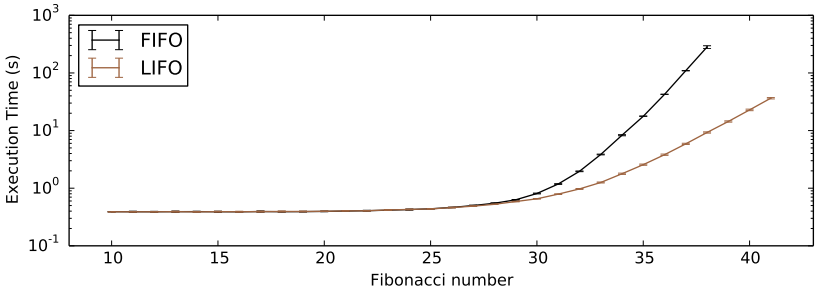


Fig. 2: Performance comparison of FIFO queue(BFS) vs LIFO queue(DFS) on the GPU backend execution of the Fibonacci program with different argument values ranging from 10 to 40.

for most number of spawns. The optimal number of spawns is always 8. These results are specific for this program, but represent how performance depends on these two parameters.

Figure 4 compares three backends for the execution of the two programs: Fibonacci and Integral. We use three backends to perform the evaluation: a multicore CPU backend, using the Æminium Runtime and 24 hyper-threads; the GPU backend using our approach; and the Hybrid approach that uses both backends. The hybrid approach does not improve the performance over the best platform because of the irregularity of the program, leaving a large critical-path on one of the platforms. It is possible to notice that after a certain workload (Fibonacci(18) and Integral(10^4)), the GPU platform is faster than the CPU backend. Smaller inputs do not contain parallelism enough to justify the overheads in GPU memory copies. However, for larger workloads we can have speedups up to 2.2 times compared to 24 threads on the CPU.

These speedups validate that our model can efficiently execute Fork-Join programs on the GPU, and represents an advancement over previous work, which could not achieve speedups over the serial version [8].

6 Related Work

The Fork-Work-Join is based on the Fork-Join model introduced in Cilk [6] and later implemented in Java [7]. The proposed model is an extension to the former model that decouples the work from the forking and joining, allowing for an efficient GPU translation. All the existing work for Fork-Join models on multicore CPUs can also be applied to this model. CPU focused languages such as OpenMP [12] and OmpSs [13] has been targeting GPUs as a backend, providing an higher-level language than CUDA or OpenCL. Despite using a Fork-Join model internally on the CPU, only data-parallelism is translated to the GPU.

Xkaapi [14] uses a Work-Stealing approach to schedule computations on multi-GPUs taking advantage of concurrent memory copies and kernel schedul-

10

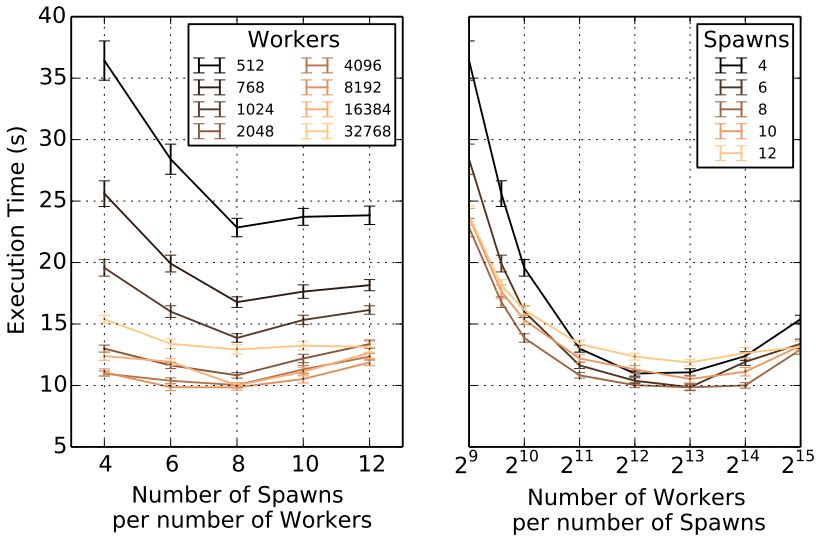


Fig. 3: Comparison of the effect of the number of GPU workers and the maximum number of spawns per GPU worker on the Fibonacci program (with argument 40) and using a LIFO queue.

ing. However it considers each GPU as a single worker, while our approach balances the load across GPU workers. There has been another study on balancing the irregular workload of a raytracer on GPUs [15]. However, it shows several possible load-balancing techniques such as work sharing, work stealing and blocking queues. Another approach is the *finish-async* model of X10 implemented on the GPU using a work-stealing approach inside the GPU [16]. The proposed model has the advantage of storing the deque of work on the GPU, known to have limited amount of memory compared to the host. Our approach limits the storage for the arguments of the computation for the general model, and it avoids that storage on the ranged model.

In version 3.1, NVIDIA has introduced recursion in CUDA kernels [1], but there are several limitations: there cannot be memory allocations in recursive kernels, and the user has to choose between parallel recursion and memory consistency. These limitations make this model unsuitable for all Fork-Join recursion tasks.

OpenCLunk [8] also implements the Fork-Join model of Cilk on the GPU, which has a similar approach to our general model, but since it does not have a separated working phase, it cannot perform the same optimizations, resulting in a slower kernel, without any speedup when compared to a single-threaded version, while our approach outperforms a multicore version. The runtime system of HDR [17] implements a work-sharing system based on the workgroup memory model in OpenCL. The HDR runtime optimizes head and tail recursions, but does not efficiently balances highly irregular programs. Kite [18] implements a conversion of Continuation Passing Style (CPS) to OpenCL using the Persistent Thread model. This approach is similar to ours in the sense that it also uses a

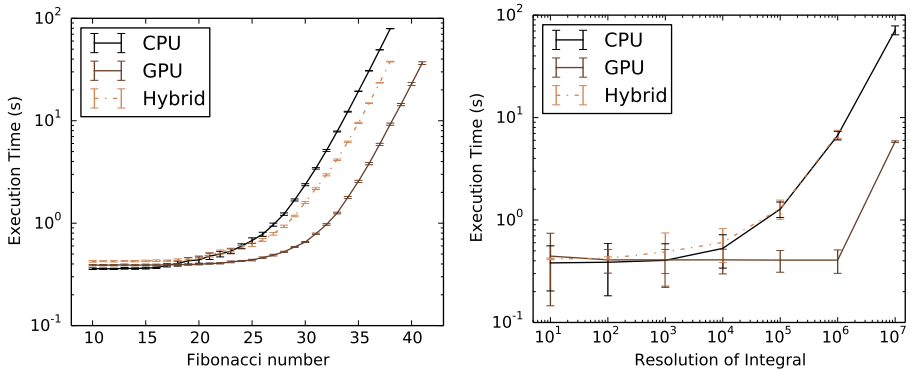


Fig. 4: Comparison of different backends (CPU, GPU and Hybrid) on two different applications: Fibonacci (a) and Integral (b) with different inputs.

translation phase, but differs since we do not use a work-stealing approach, but rather a general pass for subdividing work, a different kind of work-sharing. The runtime memory structures used to maintain the program state are similar, but Kite has more overhead since it needs to save continuations and Fork-Work-Join only needs to store whether the computation is finished or not.

Our approach allows to express any Fork-Join program on the GPU, like OpenCLunk, but in a way that it can be efficiently executed on the GPU, with speedups over both the serial version and a 24-threaded version.

7 Conclusions and Future Work

We have presented a model to express Fork-Join computations that can be efficiently executed on the GPU. The Fork-Work-Join model splits the generation of more work and the split, in order to perform multiple splits to achieve the massive parallelism required by the GPU, without much overhead.

We have implemented this model and evaluated it using two benchmarks. The two applications are traditional Fork-Join applications, and they are irregularly balanced, which makes them unsuitable for existing Map-Reduce frameworks which lack the load-balancing and recursion. We were able to achieve better performance (up to 2.2x speedup) on the GPU than on a 12-core CPU, unlike previous work [8] in which the GPU was slower than the single-threaded program.

This work can be further evaluated with real-world programs, which was out of the scope of this paper, and it could be complemented with an automatic system to select the best parameters for a given program. Another aspect left to explore is how to combine nested Fork-Join operations.

Acknowledgments This work was supported by the LaSIGE Research Unit (UID/CEC/00408/2013).

Adaptação Guiada por Políticas de Sistemas Tolerantes a Faltas Bizatinas

Miguel Pasadinhas, Daniel Porto, Antónia Lopes, and Luís Rodrigues

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{miguel.pasadinhas, danielporto, ler}@tecnico.ulisboa.pt
LASIGE, Faculdade de Ciências, Universidade de Lisboa
malopes@ciencias.ulisboa.pt

Resumo Ataques maliciosos, erros de software ou até mesmo enganos de operadores podem causar desvios arbitrários do comportamento esperado dos sistemas. A tolerância a faltas bizantinas (BFT) engloba um conjunto de técnicas que tornam os sistemas robustos na presença de faltas arbitrárias. Vários protocolos BFT foram propostos, cada um otimizado para diferentes condições operacionais e geralmente com fraco desempenho fora dessas condições. Para colmatar esse problema, propomos o Policaby, um gestor de adaptação capaz de executar políticas de adaptação que guiam o sistema num caminho de conformidade com os seus objetivos. Estas políticas são descritas numa linguagem de alto nível, com a expressividade necessária para capturar as adaptações desejáveis neste tipo de sistemas.

1 Introdução

As faltas arbitrárias, também designadas por faltas Bizantinas [11], capturam desvios arbitrários ao comportamento esperado de um componente, sejam estes derivados de causas naturais ou resultantes de ataques premeditados por agentes que pretendem interferir na operação de um dado sistema. Pela sua abrangência, os mecanismos de tolerância a faltas Bizantinas (BFT), têm sido alvo de um interesse crescente nas últimas décadas.

Infelizmente, apesar do elevado número de protocolos que foram propostos para o efeito, não existe um protocolo BFT capaz de superar qualquer um dos restantes em qualquer ambiente de execução. Na verdade, cada solução está otimizada para cenários particulares e pode ter um desempenho fraco fora desses cenários. Por exemplo, o Zyzzyva [10] é um protocolo BFT que está otimizado para o cenário em que rede é estável, mas que necessita de executar um processo de recuperação moroso caso se suspeite que um nó especial – o líder – tenha falhado.

Por esse motivo têm sido propostos sistemas adaptativos capazes de alterar o comportamento dos sistemas tolerantes a faltas Bizantinas de acordo com as condições operacionais observadas. Trabalhos como o Aliph [2] e o Adapt [3] mostraram que a adaptação dinâmica pode melhorar o desempenho dos sistemas BFT. Contudo estes sistemas possuem um suporte muito limitado para expressar

as políticas de adaptação. Uma política de adaptação é uma especificação das adaptações que podem ser realizadas, juntamente com um conjunto de informações que caracterizam os objetivos do sistema e que guiam o processo de seleção das melhores adaptações a realizar. O Aliph implementa uma única política de adaptação estática e plasmada no código fonte do próprio sistema, não prevendo qualquer tipo de ajuste à mesma. O Adapt implementa também uma única política que consiste em abortar o protocolo BFT ativo quando as condições de execução mudam e substitui-o por um outro protocolo que se comporta melhor nessas condições. A escolha do novo protocolo é guiada pelas preferências do utilizador, contudo essa configuração é complicada e pouco intuitiva.

Apesar do número de trabalhos que aborda a adaptação dinâmica de protocolos Bizantinos ser reduzido, a literatura que descreve sistemas adaptativos em contextos distintos é bastante rica [5,7,13]. É pois fácil encontrar na bibliografia exemplos de sistemas adaptativos com mecanismos e linguagens de definição de políticas mais expressivos e flexíveis do que os suportados pelo Aliph ou Adapt. Infelizmente, a maioria desses sistemas oferece um suporte limitado para gerir sistemas replicados, dos quais os sistemas BFT são um caso particular. Em particular, verificamos que algumas das adaptações mais relevantes em aplicações BFT são difíceis de expressar nos sistemas existentes.

Neste contexto, propomos neste artigo o Policaby, um gestor de adaptação que suporta uma nova linguagem para expressar políticas de adaptação de sistema BFT. O Policaby é, ele próprio, tolerante a faltas bizantinas, executando-se em múltiplas réplicas que se coordenam para assegurar a coerência mútua. A linguagem usada para expressar as políticas de adaptação tem primitivas desenhadas a pensar nas especificidades dos sistemas replicados (como os sistemas BFT), contudo é uma linguagem genérica pelo que o Policaby pode também ser usado como gestor de adaptação de sistemas sem requisitos de tolerância a faltas.

2 Trabalho Relacionado

Após os protocolos BFT inicialmente propostos em [11], foram desenhados vários protocolos mais eficientes e capazes de funcionar numa gama mais abrangente de condições [6,10,8,1]. Infelizmente, cada protocolo apresenta um desempenho mais favorável em condições de execução especificas, não existindo um protocolo capaz de superar todos os restantes em todas os ambientes de execução. Por este motivo, tem sido feito em esforço em desenvolver sistemas BFT adaptativos, capazes de se adaptar ao ambiente de execução verificado. A reconfiguração dinâmica de um único protocolo, ou a comutação em tempo de execução entre diferentes protocolos, são exemplos de adaptações que podem ser concretizadas.

Um dos primeiros sistemas capaz de alterar o protocolo BFT face a variações nas condições de operação foi o Aliph [2]. No entanto, este sistema apenas concretiza uma única política de adaptação, definida de forma estática, que consiste em substituir um protocolo por outro sempre que as condições de operação assim o justificam. Neste sistema, os protocolos suportados são organizados num anel

e troca de protocolos segue a ordem definida na construção deste anel, o seja, um protocolo quando é desativado, é sempre substituído pelo protocolo seguinte no anel. As condições que definem quando cada protocolo deve ser desativado estão plasmadas estaticamente no código fonte, e capturam o conhecimento do programador sobre o desempenho de cada um dos protocolos. Assim, não existe no Aliph suporte para expressar novas políticas de adaptação do sistema – existe apenas uma que está embutida no código. Para além disso, o Aliph é um sistema monolítico, sem um gestor de adaptação externo ao sistema que é adaptado – é o próprio sistema que decide as próprias adaptações.

O Adapt [3] é um sistema BFT adaptativo que recorre a uma arquitetura de três camadas, nomeadamente: o sistema que deve ser adaptado (BFTS), um sistema de eventos (ES) que monitoriza continuamente o BFTS e um sistema de controlo de qualidade (QCS) que recebe dados do ES e decide as adaptações. Esta arquitetura é inspirada na malha de controlo MAPE-K [9], usada recorrentemente em sistemas adaptativos não monolíticos. Apesar do BFTS ser tolerante a faltas bizantinas, nem o ES nem o QCS o são. À semelhança do Aliph, as adaptações suportadas pelo Adapt são exclusivamente a troca de um protocolo BFT por outro mas, ao contrário do Aliph, a sequência de protocolos escolhida não é fixa. O Adapt introduziu um passo intermédio no qual decide qual o melhor protocolo para as condições de execução detetadas pelo ES. Para realizar essa decisão o Adapt baseia-se em métricas chave (monitorizadas pelo ES) que se consideram ser capazes de caracterizar o estado do sistema. Exemplos destas métricas são a latência e o débito do sistema. O processo de decisão passa pela construção de uma matriz que representa o desempenho esperado de cada protocolo (nas linhas) face a cada métrica chave (nas colunas). Cada entrada da matriz é calculada com o auxilio de uma função de previsão previamente treinada. O Adapt possui também um vetor que associa um peso a cada métrica chave, peso este que é fornecido pelo administrador do sistema. Assim, da média ponderada do desempenho esperado de cada protocolo resulta num valor que é considerado a utilidade desse protocolo. O protocolo com maior utilidade é escolhido para executar. As políticas de adaptação do Adapt baseiam-se, portanto, num vetor de pesos, no qual um administrador de sistema pode atribuir maior importância a determinadas métricas chave em detrimento de outras. Apesar de ser uma melhoria face às políticas não configuráveis do Aliph, este mecanismo não é muito expressivo. Dada a proliferação de fatores, é difícil prever o impacto das políticas que resultam de diferentes ponderações dos fatores, por exemplo, é difícil estimar o efeito de dar um peso duas vezes maior à latência face ao débito.

O ByTAM [14] é um gestor de adaptação que segue também uma arquitetura de três camadas, semelhante ao Adapt – possui um sistema gerido, um sistema de monitorização e um gestor de adaptação. O ByTAM tornou o sistema de monitorização e o gestor de adaptação tolerantes a faltas bizantinas, sendo uma melhoria face ao Adapt. Ao contrário dos sistemas vistos anteriormente, o ByTAM permite a especificação de adaptações genéricas, não estando limitado à troca de protocolos. As políticas suportadas pelo ByTAM são expressas através de um conjunto de regras na forma de evento-condição-ação, sendo que os obje-

tivos que a política pretende atingir ficam codificados apenas de forma implícita nestas regras. Para além disso, a especificação destas regras requer a codificação de uma interface em Java, pelo que as políticas têm que ser compiladas juntamente com o gestor de adaptação. Isto significa que podem ser especificadas políticas extremamente complexas (uma vez que se pode escrever código Java arbitrário) mas a um nível de abstração pouco apropriado. Para escrever políticas simples o utilizador tem que lidar com toda a complexidade do Java. Este facto combinado com a inexistência de uma API própria que facilite a escrita das políticas tornam este mecanismo difícil de utilizar na prática.

Existem na literatura várias técnicas e linguagens para especificação de políticas de adaptação mais flexíveis e expressivas do que as apresentadas nos sistemas acima. A linguagem Stitch [5,7] baseia-se na especificação de táticas (primitivas de adaptação que possuem ações e os seus impactos esperados sobre algumas métricas chave) que têm a particularidade de suportar impactos não deterministas. O não-determinismo é capturado especificando os impactos como cadeias de Markov em tempo discreto. Estratégias de adaptação são árvores de decisão em que cada ramo está protegido por uma guarda e os nós representam táticas. Estas árvores pretendem representar o processo de decisão de um administrador de sistema quando manualmente adapta um sistema. Para decidir qual a melhor estratégia a executar, o Stitch baseia-se em na média ponderada (com pesos fornecidos pelo utilizador) da utilidade de várias métricas chave. A utilidade de uma métrica é um valor entre 0 e 1, calculado por funções especificadas pelo utilizador, que mapeia valores da métrica em valores de utilidade. Este passo intermédia de calcular a utilidade das métricas permite lidar com a descrepancia nas gamas dos valores das várias métricas. A linguagem proposta por Liliana et al. [13] também se baseia na definição da adaptações que têm ações e impactos esperados. Ao contrário dos anteriores, este sistema baseia-se numa lista ordenada de objetivos para escolher a melhor adaptação a executar. Este mecanismo torna-se mais intuitivo de configurar uma vez que não é necessário descobrir quais os pesos necessários para mapear os objetivos de negócio. Em vez disso existe um mapeamento muito mais direto entre os objetivos de negocio e os objetivos especificados na linguagem.

Apesar da elevada expressividade fornecida por estas linguagens, existem algumas adaptações que se revelam difíceis ou mesmo impossíveis de expressar nestas linguagens. A título de exemplo, estas linguagens não suportam especificar uma adaptação que seleciona a réplica mais apta para ser líder do protocolo BFT. Como vamos ver mais à frente, expressar este tipo de políticas no Policaby é intuitivo devido ao mecanismo de parametrização das adaptações.

3 Linguagem

Parte fulcral do Policaby é a linguagem usada para especificar as políticas de adaptação. A linguagem foi desenhada com o objetivo de especificar políticas de adaptação para sistemas replicados, tais como os sistemas BFT. Como vimos, os gestores de adaptação existentes para adaptação de sistemas BFT não oferecem

suporte para expressar políticas de adaptação e linguagens como as descritas em [13,7], apesar de muito flexíveis, não permitem capturar de forma simples algumas adaptações relevantes em sistemas BFT. Por questões de espaço, nesta secção vamos apresentar um subconjunto da linguagem. Uma apresentação sistemática e detalhada da linguagem pode ser encontrada em [12].

3.1 Adaptações

A maioria do protocolos BFT recorrem à eleição de um líder e uma adaptação relevante consiste em trocar o processo que assume este papel. Por exemplo, se um sistema BFT estiver a usar o protocolo Zyzzyva, em regime estável a comunicação é feita apenas entre o líder e as restantes réplicas (não existindo comunicação direta entre duas réplicas não líderes). Assim, ao selecionar como líder a réplica que tem menor latência para a restantes réplicas é possível reduzir o tempo necessário para conseguir a coordenação entre os processos (que designamos por *t_ coord*). Vamos assumir que esse tempo de coordenação é inversamente proporcional à latência que o líder tem para as restantes réplicas (uma simplificação que fazemos para conveniência da exposição). Se pretendêssemos especificar esta adaptação com os mecanismos descritos em [7,5] poderíamos usar seguinte política:

```
1  define P = p: Protocolo | p.zyzzyva = true & p.activo = true
2  define l = oneOf{r:Replica | r.lider = true}
3  define novoLider = oneOf{r:Replica | r.lider = false &
4    forall t:Replica | r.latencia < t.latencia}
5
6  impactmodel mudarLider
7    size(P) > 0 -> {
8         [0.9] {novoLider.lider' = true & l.lider' = false &
9                t_coord' = t_coord * novoLider.latencia / l.latencia} +
10        [0.1] {novoLider.lider' = false & l.lider' = true}}
```

Listagem 1.1. Adaptação de mudança de líder especificada com os mecanismos descritos em [7,5].

Esta especificação captura o impacto esperado de realizar a adaptação que consistem em mudar o líder. A linguagem permite capturar fatores não deterministas, nomeadamente a possibilidade da adaptação falhar e o líder permanecer o mesmo (nesta caso, com uma probabilidade estimada de 0.1). Nesta especificação, a política define explicitamente qual o critério para escolher o novo líder; isto é feito na linha 3, onde é definido que o novo líder será aquele que possuir uma menor latência para os restantes. No entanto, considerando que o impacto de escolher um determinado líder já está capturado na política, não deveria ser necessário explicitar o critério pata escolher o líder. Ao invés, a política deveria definir um objetivo de negócio (por exemplo, reduzir a latência), e o motor de execução da política deveria usar os impactos para escolher o melhor líder. Uma forma alternativa de atacar este problema seria escrever n adaptações, em que cada uma representava o impacto de mudar o líder para a réplica i, em que n é o número de réplicas existentes e $i \in \{1, ..., n\}$. Desta forma o líder seria escolhido implicitamente pela escolha da melhor adaptação.

O trabalho de Rosa et al. [13] introduz o conceito de adaptação de nó (*node adaptation*), uma adaptação que atua sobre um nó específico de um sistema distribuído. Em execução, cada nó vai ser testado, permitindo assim descrever com apenas uma adaptação o impacto de realizar essa adaptação tendo cada nó como alvo. Infelizmente as únicas adaptações de nó permitidas são adição e remoção de componentes de um nó específico, não dando possibilidade para a mudança de uma configuração global do sistema (como uma mudança de líder) com base numa adaptação de nó.

A linguagem que desenvolvemos para o Policaby evita estas limitações. Apresentamos agora uma possível especificação da adaptação de mudança de líder no Policaby:

```
1  Adaptation mudarLider:
2      Input:
3          r: Replica
4      Requires:
5          System.lider != r
6          System.protocolo = Zyzzyva
7      Impacts:
8          [0.90]:
9            System.t_coord *= r.latencia / System.lider.latencia
10           System.lider = r
11         [0.10]:
12           System.lider = System.lider
```

Listagem 1.2. Especificação de uma adaptação de mudança de líder no Policaby.

A especificação de uma adaptação no nosso sistema possui uma sintaxe que separa claramente as condições em que a adaptação é aplicável (com a primitiva *Requires*) e os impactos esperados (com a primitiva *Impacts*). Tal como em [7,5], continua a ser possível indicar que uma adaptação pode ter diferentes resultados, cada um com uma probabilidade esperada de acontecer. Uma vez que o impacto da adaptação depende de qual a réplica escolhida, e não queremos à partida explicitar o algoritmo para seleção do líder, introduzimos na linguagem um mecanismo de parametrização das adaptações, que generaliza os mecanismos propostos em [13]. As adaptações podem especificar parâmetros formais aos quais vão ser associadas entidades concretas durante a execução. No exemplo, a adaptação é parametrizada por um elemento r do tipo *Replica* – a réplica que será o novo líder. Quando o Policaby for decidir que adaptação mais beneficia o sistema, esta adaptação será instanciada tantas vezes quanto o número de réplicas existentes no momento. A avaliação dos impactos de cada instância resulta num estado diferente pois os impactos são especificados em função do parâmetro formal r. As adaptações podem especificar tantos parâmetros formais quanto os necessários. Por exemplo, se alguma adaptação necessitar de um par de réplicas, pode especificar dois parâmetros formais do tipo Replica. É importante referir que apenas é possível parametrizar adaptações com tipos componente (descritos na Secção 3.2) que são especificados pelo utilizador. Associado a cada adaptação existe um *script* que é o código usado para realizar a adaptação. Quando o Policaby decide uma adaptação, o *script* associado a essa adaptação é executado. O *script* é especificado usando a linguagem de programação Groovy que tem acesso às *API* fornecidas pelo Policaby para facilitar a execução das adaptações.

3.2 Componentes e Instâncias

Como mostrado na listagem 1.2, as adaptações podem ser parametrizadas com parâmetros formais de tipos específicos. Esses tipos correspondem a componentes na linguagem. As componentes servem fundamentalmente dois objetivos: (1) especificação de tipos cujas instâncias podem variar ao longo do tempo e (2) especificação de tipos enumerados. Como exemplo do primeiro objetivo temos o conceito de réplica. As réplicas concretas que existem no sistema variam ao longo do tempo, sendo necessário existir um tipo para as representar. Como exemplo do segundo objetivo temos protocolos BFT. Podemos definir Protocolo como uma enumeração cujas entidades são os protocolos concretos. Na listagem 1.3 podemos ver a definição de uma componente *Replica* com atributos descritos como ou mensuráveis ou não mensuráveis. Os atributos mensuráveis têm os seus valores fornecidos por um sistema de monitorização. Vamos explorar a interação com este sistema na Secção 4. Por sua vez, os atributos não mensuráveis apenas mudam de valor como resultado de uma adaptação. Assim, devemos usar atributos mensuráveis para especificar atributos cujo valor está fora do controlo do Policaby e que devem portanto ser medidos – como a latência das réplicas – e atributos não mensuráveis sobre parâmetros sobre os quais temos controlo total. No exemplo, o custo de uma réplica foi especificado como não mensurável, significando que o custo de cada réplica é conhecido e não varia ao longo do tempo. Se imaginarmos um cenário em que o custo das réplicas varia fora do controlo do Policaby, então deveria ser usado um atributo mensurável.

```
1 Enum Protocolo Instances Zyzzyva, PBFT, Aardvark
2
3 Component Replica:
4     Params:
5         mensurable latencia: number
6         mensurable debito: number
7         custo: number
```

Listagem 1.3. Definição de Componentes e Instâncias no Policaby.

O conjunto de réplicas dos sistemas replicados pode variar ao longo do tempo (por exemplo, como resultado de adaptações de adição/remoção de réplicas). Como resposta a esse requisito, podem ser criadas e destruídas instâncias dos tipos componentes em tempo de execução. Existe, para esse efeito, uma *API* que pode ser usada nos *scripts* de adaptação, que permite criar e destruir instâncias de qualquer componente.

Para além dos atributos associados a componentes, existe também um conjunto de atributos globais do sistema. Os tipos desses atributos podem ser componentes, permitindo assim caracterizar a configuração atual do sistema (por exemplo notando qual a réplica que é líder ou qual o protocolo ativo).

3.3 Objetivos da Adaptação

Como é comum em gestores de adaptação, assume-se que os objetivos do sistema gerido podem ser especificados em função de um conjunto de indicadores de desempenho alvo (no inglês *key performance indicator*, KPI). No Policaby, todos

os atributos das componentes e atributos globais são considerados KPIs. Isto significa que tanto os impactos das adaptações como o objetivo do sistema gerido são especificados em função desses atributos. Por conveniência, os KPIs podem ser agregados em KPIs compostos. A título de exemplo, podemos especificar um KPI *custo_ total* como sendo a soma do custo de cada réplica: `KPI custo_total Is Sum Replica.custo`.

À semelhança da linguagem proposta em [13], os objetivos da adaptação são especificados recorrendo a uma lista de sub-objetivos, ordenada por ordem de importância. Esta aproximação é mais expressiva do que soluções que apenas permitem expressar um único objetivo, que consiste em maximizar uma função de utilidade, tipicamente uma média ponderada da distância a valores alvo para cada KPI. Por exemplo, podemos especificar que o custo total das réplicas deve ser inferior a 2 Euro/hora – `Goal custo_total < 2 Confidence 0.9` – ou que desejamos minimizar o tempo despendido a coordenar pedidos – `Goal Minimize System.t_coord`. Como assumimos um modelo não-determinístico para os impactos das adaptações, alguns objetivos (ditos *exatos*, isto é, que especificam uma condição de aceitação) podem especificar o grau de confiança mínimo para ser considerado cumprido. No exemplo do custo total podemos ver que especificámos uma confiança mínima de 0.9 que o objetivo seja considerado cumprido. Assim, apenas se considera que uma adaptação cumpre este objetivo se a soma das probabilidades dos ramos que cumprem o objetivo for superior a 0.9. Por outro lado os objetivos de otimização (maximização e minimização de expressões) recorrem à média do valor obtido em cada ramo ponderado pela sua probabilidade. Uma característica vantajosa da abordagem de objetivos ordenados é que fornece uma degradação graciosa caso não seja possível cumprir todos os objetivos, pois aqueles que são considerados mais relevantes são garantidos em primeiro lugar.

4 Policaby

Nesta secção descrevemos o Policaby. Em primeiro lugar descrevemos a arquitetura de três camadas assumida; de seguida apresentamos o algoritmo de decisão usado pelo motor de avaliação das políticas.

4.1 Arquitetura

O Policaby é um gestor de adaptação replicado, tolerante a faltas Bizantinas, capaz de decidir adaptações que guiam um Sistema Gerido (distinto do Policaby) num caminho de conformidade com os seus objetivos. Como requisito para tomar essas decisões, deve existir um Sistema de Monitorização capaz de monitorizar algumas métricas chave sobre o Sistema Gerido. Esse conhecimento gerado pelo Sistema de Monitorização caracteriza o ambiente de execução do Sistema Gerido, permitindo ao Policaby selecionar as melhores adaptações face a esse ambiente.

O fluxo de informação é o seguinte: o Sistema de Monitorização usa sensores para recolher métricas chave sobre o Sistema Gerido; essas métricas são tratadas pelo Sistema de Monitorização e enviadas para o Policaby; com esse

conhecimento, o Policaby decide se existe uma adaptação que melhore o desempenho do sistema; por fim, caso exista uma adaptação a ser executada, essa decisão é comunicada ao Sistema Gerido. As métricas enviadas pelo Sistema de Monitorização são totalmente ordenadas pelo BFT-SMaRt [4], uma biblioteca para a construção de serviços tolerantes a faltas Bizantinas. O BFT-SMaRt implementa um protocolo de tolerância a faltas bizantinas e garante que todas as réplicas recebem as mensagens em ordem total. Por se tratar de um serviço construido em cima do BFT-SMaRt, o Policaby é tolerante a faltas bizantinas. O Policaby executa o algoritmo de decisão de adaptações periodicamente, sendo que essa execução é despoletada por uma mensagem vinda de um temporizador (proporcionado pelo Sistema de Monitorização). Novamente, esta mensagem é totalmente ordenada pelo BFT-SMaRt. A ordem total de entrega de mensagens garantida pelo BFT-SMaRt garante que todas as réplicas do Policaby recebem todas as mensagens pela mesma ordem e como resultado todas as réplicas tomam a mesma decisão.

4.2 Motor de Avaliação das Políticas

O Policaby é composto por duas partes fundamentais: (1) um analisador que transforma especificações escritas na linguagem descrita na Secção 3 em objetos internos e (2) um motor capaz de usar esses objetos para decidir adaptações que guiam o sistema gerido ao encontro dos seus objetivos. O motor corre periodicamente e resulta na execução de uma adaptação. Existe sempre no sistema uma adaptação *NOP* que não realiza qualquer ação. Isto permite que, se nenhuma das restantes adaptações melhorar o sistema, nenhuma ação seja tomada.

O algoritmo de decisão do motor de avaliação das políticas é o seguinte. Em primeiro lugar, é criado um conjunto S contendo a adaptação *NOP*. Depois, cada adaptação é instanciada com base dos seus parâmetros formais e cada instância é colocada no conjunto S. A instanciação de uma adaptação corresponde à associação dos seus parâmetros formais com instâncias concretas das componentes associadas. Por exemplo, a adaptação *mudarLider* da listagem 1.2 seria instanciada n vezes, uma para cada réplica existente. De notar que se uma adaptação possuir mais do que um parâmetro formal, todas as combinações são geradas. No final deste passo, todas as combinações possíveis de instâncias de adaptações estão em S. Neste ponto, são verificadas as condições de aplicação das adaptações (especificadas com a primitiva *Requires*) e as instâncias de adaptações que não cumpram todas as condições são removidas de S. As adaptações presentes em S correspondem agora a todas as instâncias de adaptações válidas para a decisão. Para cada uma é gerado um estado esperado, avaliando os impactos da adaptação. Resta apenas verificar qual o estado esperado mais favorável de acordo com os objetivos. Assim, começando pelo objetivo mais prioritário (que é o que foi definido mais acima nas políticas), o conjunto S é validado contra cada objetivo. Os elementos de S que não estejam em conformidade com o objetivo são removidos, até que S contenha apenas um elemento ou os objetivos acabem. No final destes testes, se S contiver apenas um elemento, então essa é a adaptação mais vantajosa. Caso S contenha vários elementos, considera-se que

todas as adaptações presentes são equivalentes, e uma aleatória é escolhida. Caso nenhuma adaptação consiga realizar um determinado objetivo, esse objetivo é descartado e todas as adaptações presentes em S passam automaticamente ao próximo objetivo. Desta forma os restantes objetivos podem ser usados como fator de desempate.

5 Avaliação

O Policaby permite especificar uma gama de políticas muito mais abrangente do que sistemas como o Adapt ou o Aliph. Os mecanismos como parametrização de adaptações, não-determinismo associado às adaptações e adaptação baseada em objetivos ordenados dotam a linguagem com a capacidade de especificar políticas de adaptação que não são permitidas nesses sistemas. Como contrapartida, o processo de escolha de adaptações torna-se computacionalmente mais exigente.

A título de exemplo, uma adaptação que seja instanciada 10 vezes e que, por via do não-determinismo, possua dois ramos alternativos, corresponde a 20 estados possíveis com uma única adaptação. A escalabilidade do sistema torna-se portanto um aspeto merecedor de atenção.

Para aferir a escalabilidade do sistema, medimos o tempo desde que o Policaby inicia o processo de decisão até que decide qual a melhor adaptação, numa réplica com processador Intel Core i7 860 (2.80GHz), com 8GB de RAM DDR, disco SSD de 280GB e sistema operativo Debian 8.5. Para cada configuração testada, o tempo apresentado corresponde ao valor médio de 100 experiências.

Nesta experiência a política tem n adaptações (variando n nas várias configurações testadas) com dois ramos (com probabilidades 0.3 e 0.7) e cada ramo especifica três funções de impacto. Para além disso, esta adaptação é parametrizada por duas componentes, das quais existiam 4 instâncias, resultando em cada adaptação ser instanciada 16 vezes. Metade dos objetivos usados são exatos e a outra metade de otimização. À exceção de uma configuração, todas as instâncias de adaptação passavam todos os objetivos. Isto corresponde ao pior caso para o Policaby, uma vez que nenhum objetivo filtrou instâncias de adaptação e consequentemente, todas as instâncias de adaptação foram ser testadas contra todos os objetivos.

Na Figura 1 podemos ver os resultados experimentais. O eixo horizontal codifica o número de instâncias de adaptação testadas e o eixo vertical o tempo de execução em milissegundos. Cada série com linha cheia corresponde a um número diferente de objetivos. Podemos ver que, para um número fixo de objetivos, aumentar o número de instâncias de adaptação aumenta o tempo de decisão linearmente. O mesmo se verifica com o aumento do número de objetivos (com número de instâncias de adaptação constante). A série que usa 0 objetivos permite entender o tempo gasto para instanciar as adaptações e gerar os estados resultantes de avaliar as funções de impacto. Assim, conseguimos entender que maior parte do tempo do processo de decisão é gasto na avaliação dos objetivos. Fizemos também um teste com 32 objetivos que rejeitavam (em média) 15% das instâncias de adaptação testadas. Este caso corresponde à série a tracejado

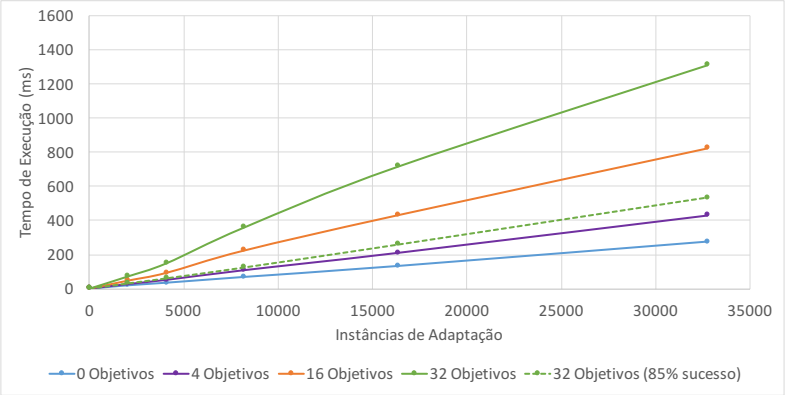


Figura 1. Tempo de decisão de uma réplica.

no gráfico e simula um caso de uso mais real, em que nem todas as adaptações conseguem cumprir todos os objetivos. Comparando os resultados de ambas as séries com 32 objetivos, podemos ver que mesmo um taxa de rejeição baixa de 15% permite melhorias significativas no tempo de decisão (superior a 50%), conseguindo o Policaby decidir uma adaptação entre mais de 32.000, tendo em conta 32 objetivos, em cerca de meio segundo.

6 Conclusões e Trabalho Futuro

Neste artigo apresentámos o Policaby, um gestor de adaptação robusto com uma nova linguagem para expressar políticas de adaptação. Este linguagem suporta especificação de adaptações parametrizadas, permitindo assim escrever adaptações que implicitamente selecionam as melhores instâncias como argumentos. A linguagem suporta também um mecanismo que permite especificar diversos ramos (com probabilidade associada) para uma adaptação, permitindo assim lidar com a incerteza do impacto da adaptação. Para selecionar os objetivos o Policaby baseia-se numa lista ordenada de objetivos que o sistema gerido deve cumprir. Tanto quanto é do nosso conhecimento, o Policaby é o primeiro gestor de adaptação que lida com não-determinismo das adaptações capaz de escolher adaptações com base numa lista de objetivos. Em trabalhos futuros devem ser explorados mecanismos que permitam ao Policaby escolher uma sequência de adaptações em vez de uma única adaptação.

Agradecimentos: Este trabalho for parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projectos com referências PTDC/ EEI-SCR/ 1741/ 2014 (Abyss) e UID/ CEC/ 50021/ 2013.

Support for Automatic Refactoring of Business Logic

Tiago B. Fernandes[1], António Nestor Ribeiro[2], David V. Nunes[3], Hugo R. Lourenço[3], and Luiz C. Santos[3]

[1] Dep. Informática Universidade do Minho
[2] Dep. Informática Universidade do Minho/Haslab - InescTec
[3] OutSystems, Inc.

Abstract. Software's structure profoundly affects its development and maintenance costs. Poor software's structure may lead to well-known design flaws, such as large modules or long methods. A possible approach to reduce a module's complexity is the Extract Method refactoring technique. This technique allows the decomposition of a large and complex method into smaller and simpler ones, while reducing the original method's size and improving its readability and comprehension.
The OutSystems platform is a low-code platform that allows the development of web and mobile applications that rely on a set of visual Domain-Specific Languages (DSLs). Even low-code languages when improperly used can lead to software that has maintenance issues like long methods.
Thus, the purpose of this paper is to present the research and development done to provide the OutSystems platform with a tool that automatically suggests Extract Method refactoring opportunities. The research combines program slicing techniques with code complexity metrics to calculate the best refactoring opportunities that preserve programs' functionality.
The proposed approach was tested on typical OutSystems apps and was shown to be able to reduce the overall applications' complexity.

Keywords: Refactoring · Program Slicing · Code Complexity Metrics · OutSystems

1 Introduction

The complexity, size and inconsistency of software demand increased maintenance costs and excessive investment of time for developing and managing it. Several studies point out causes that limit any system's functionality and development: Gill and Kemerer [1991] referred constraints caused by the high complexity degree; Banker et al. [1993] alerted the difficulties caused by large systems; Meyers and Binkley [2007] analyzed the problems caused by the lack of system's cohesion.

Software is constantly changed throughout its lifecycle. These changes are caused by bugs correction, new features implementation and adoption of new architectures, practices or development teams. Over time, all of these changes can lead to complex, graceless and incomprehensible modules.

Thus, the purpose of this paper is to provide the OutSystems platform with a tool that combines code complexity metrics and program slicing techniques to reduce modules' overall complexity.

1.1 Context and Objectives

OutSystems offers an alternative to traditional software development models. Its proprietary low-code platform allows rapid application delivery with minimal hand-coding. The platform along with its visual DSL help programmers to focus on the functional part of applications, and abstract themselves from implementation details, that result in an increased productivity and software quality. Despite the proven improvements provided by the OutSystems Platform, refactoring techniques may still help to maintain the balance and consistency of software projects.

The application of refactoring techniques is fundamental to moderate software complexity. Refactoring is the process of changing a software system to improve the design of the code without affecting its external behavior [Fowler et al., 2000]. However, manual refactoring is risky and thus it may imply programmers to be reluctant to its practice. Such behaviour may be due to various factors: lack of knowledge of refactoring techniques; the fact its benefits is only seen in the long term; being a complementary activity to the development of features and bug fixing; being a risky operation that may introduce bugs. Furthermore, refactoring is riskier if it is practiced in a disorganized way or without a methodical approach.

Projects developed in the OutSystems Platform may benefit from the application of some refactoring techniques, given the existence of large and complex modules. These large modules are often referred to as "Long Methods" in Object-Oriented Programming. In the OutSystems' context, an action has similar behavior to a method, and thus we refer to it as "Long Action". One of the most widely used techniques for reducing the complexity of modules is the Extract Method primitive [Fowler et al., 2000], which is characterized by extracting code from a routine to a subroutine so that the original routine's complexity reduces.

The OutSystems Integrated Development Environment (IDE), as well as other industry-established IDEs, has refactoring primitives such as the Extract Action. However, the identification of refactoring opportunities remains a process dependent on human intervention. Thus, automatic identification of refactoring opportunities is imperative, in a context where IDEs supervise and suggest changes in the code.

Refactoring applied externally and exclusively by the programmer must be assisted by the IDE itself, through the presentation of suggestions and respective benefits. The aid of IDEs in the automatic identification and application of refactoring techniques guarantees a methodical intervention which is not always guaranteed by an expert. Moreover, being the IDE the one responsible for changing the software makes the entire refactoring process faster and more efficient. The methodical application of refactoring techniques is essential, so that the system maintains its expected quality and integrity.

Thus, the purpose of this paper is to present the research and development done to provide the OutSystems platform with a tool that automatically suggests Extract Action refactoring opportunities. The research combines program slicing techniques with code complexity metrics to calculate the best refactoring opportunities that preserve programs' functionality.

2 Related Work

Given the wide complexity range of modules, it is imperative to analyze which modules are the most complex. This way, it is possible to select the most useful actions to refactor.

Complexity analysis can be addressed in two ways: quantitative analysis and structural analysis. Quantitative metrics consider program quantitative aspects such as its dimension. For example, the Lines of Code (LOC) metric measures the program's length, i.e., it counts the number of rows. This metric is often used in software development, given its simplicity to understand and apply.

On the other hand, structural metrics consider program structure, e.g., they study the number of paths of an action and the data flow carried. For instance, McCabe [1976] introduced the Cyclomatic Complexity ($CC = e - n + 2p$) metric that measures a program's structure based on the paths of the Control Flow Graph (CFG) [Allen, 1970]. Madi et al. [2013] present the metrics Total Cyclomatic Complexity (TCC) and Coupled Cyclomatic Complexity (CCC). While TCC aims at measuring the global Cyclomatic Complexity (CC) of a module, CCC intends at measuring the coupling level between modules' components.

Next, it became necessary to decompose modules into simpler ones to reduce their complexity. For that, there were studied multiple program slicing techniques that are able to extract behaviour from complex modules.

Program slicing is a technique that allows code extraction from a routine to a subroutine while reducing the complexity of the first. Slicing was originally designed to ease debugging processes [Weiser, 1981], but quickly became relevant in metrics calculation, code analysis and maintenance [Harman and Hierons, 2001, Tsantalis and Chatzigeorgiou, 2011].

Concerning runtime information, static slicing [Weiser, 1981] consists of all statements that can affect the value of a variable v and is directly associated with the total computation of a variable (complete computation slice). On the other hand, dynamic slicing [Korel and Laski, 1988] considers a specific value of the variables for a specific execution of a program to present better slices (regarding the input values). Conditioned slicing [Canfora et al., 1998] does not consider the variables' values but rather the conditions' values, i.e., whether or not a control predicate is true.

Regarding flow direction, backward slicing produces slices that contain all statements and control predicates that may affect a variable at a specific statement (slicing criterion). On the other hand, forward slicing [Bergeretti and Carry, 1985] produces slices that contain all statements and control predicates that may be affected by a variable at a specific statement.

Concerning syntax preservation, syntax-preserving slicing aims at extracting code from a routine to a subroutine by extracting statements and control predicates without changing them. On the other hand, amorphous slicing Harman and Danicic [1997] computes slices using syntactic transformations that may change the code, e.g., by changing control predicates.

Regarding slicing scope, intraprocedural slicing uses a single procedure as a boundary to compute slices, while interprocedural slicing [Horwitz et al., 1988] generates slices with respect to multiple functions calls. Intraprocedural slicing uses the Program Dependence Graph (PDG) [Ferrante et al., 1987] that represents control and data dependencies between nodes and interprocedural slicing uses the System Dependence Graph (SDG) [Horwitz et al., 1988] to represent interactions between multiple PDGs of a module.

Maruyama [2001] introduced the concept of block-based slicing to produce slices that do not use the whole procedure as region, but rather a smaller portion of the code. By limiting the slice expansion it is possible to extract a single variable in multiple ways, which result in more suggested slices.

3 The OutSystems Platform

The OutSystems Platform is a high-productivity platform intended for developing enterprise web and mobile applications. Its low-code IDE allows programmers to quickly develop applications with little effort and reduced costs resulting in a shorter Time to Market. It comprises a set of symbols, the most relevant of which are:

- The Start node indicates the beginning of the flow.
- The End node indicates the ending of the flow.
- The Server Action call node execute server side functions.
- The Aggregate node easily allows to retrieve data from a database.
- The If node executes a specific branch based on its condition.
- The For Each node iterates over a list of items.
- The Assign node which assigns a value to a variable. This node may have multiple Assignments, avoiding the creation of multiple Assigns.

Figure 1 represents, in the OutSystems platform, an adaptation done by Tsantalis and Chatzigeorgiou [2011] of a well-known refactoring example presented by Fowler et al. [2000]. An action in OutSystems is a flowchart with a single entry point and may have multiple exit points.

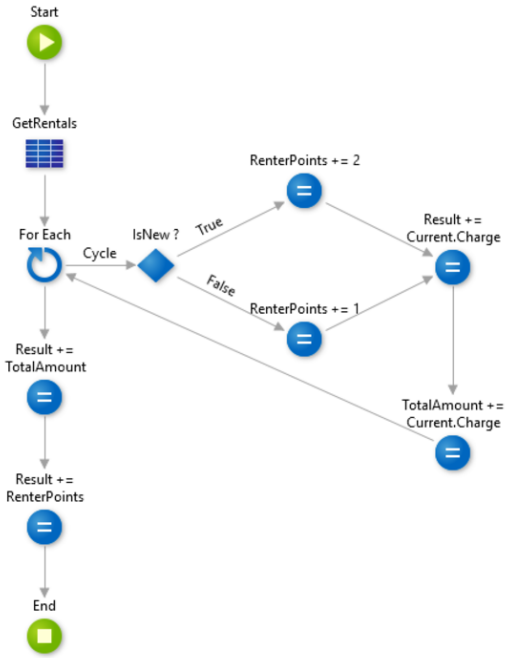


Fig. 1: Statement action in OutSystems

4 Proposed Solution

The proposed solution intends, in the first place, to evaluate the complexity of modules in order to ascertain the need of their refactoring. This step identifies the actions that are most likely to reduce their complexity after refactoring. The complexity of modules is calculated based on the number of nodes (equivalent to LOC in the OutSystems context) and the value of CC. According to Software Improvement Group (SIG), these are two relevant metrics for analyzing the quality and degree of software maintenance [Visser et al., 2016]. Both metrics are used to evaluate whether an action should be refactored by selecting the ones that would produce the best refactoring opportunities. The used threshold values were the ones defined in SIG [Visser et al., 2016]: 15 nodes for LOC and 4 units of CC.

Secondly, the proposed solution aims at reducing the complexity of modules by applying the Extract Action refactoring technique. Program slicing was the technique used to identify parts of the code that compute a common variable and then extract them to a different action.

The proposed solution is based on the Tsantalis and Chatzigeorgiou [2011] proposal and presents the following properties: (i) backward slicing, (ii) static slicing, (iii) intraprocedural slicing, (iv) syntax preserving slicing and (v) block-based slicing (see Section 2). A backward slice is computed by ignoring parts of the code that do not affect the slicing criterion, while a forward slice is computed by ignoring the statements that cannot be affected by the slicing criterion [Harman and Hierons, 2001]. Since there was no evidence of any performance increase between the two, we kept with the original definition of program slicing [Weiser, 1981] (in fact, the flow direction of the traversal should not affect slice computation). Static slicing was chosen over dynamic slicing because it is a more generic proposal as it computes all possible slices independent of user input. The use of a single PDG at a time (intraprocedural slicing) was to reduce implementation efforts. Choosing syntax preservation was an in-house decision to avoid possibly changing the original program's structure. Block-based slicing was the implemented solution as it allows a greater number of refactoring opportunities, since it promotes the extraction of variable within the multiple regions of the code. In this context, the technique is used to extract parts of the business logic from an action to another to reduce the first's complexity.

Maruyama [2001] proposed block-based slicing as a means to produce slices scoped in a block-based region. A block-based region $R(B_n)$ is a set of nodes in the PDG along with control and data dependencies between them. A block-based region starts at a specific basic block and ends at the last reachable one. A basic block B_n is a sequence of consecutive nodes in the CFG without branching and are delimited by leader nodes. A leader node is (i) the first node, (ii) a join node or (iii) a node that directly follows a branch node. The Start and the End nodes do not belong to any basic block. Figure 2 represents the basic blocks of the action presented in Figure 1.

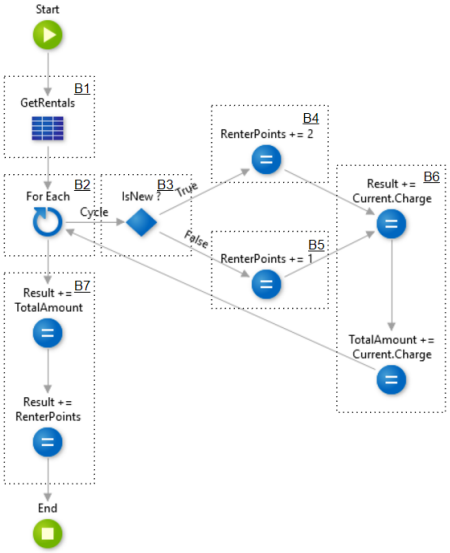


Fig. 2: Basic blocks of the CFG

Maruyama defines a block-based slice $S_B(n, v, B_n)$ as a backwards static slice that contains all statements that may affect the value of slicing criterion $C = (n, v)$ (given node n and variable v) within region $R(B_n)$ (Equation 1).

$$\begin{aligned} S_B(n, v, B_n) &= \{m \in N(G) \mid m \rightarrow^* n \in E_B(R(B_n))\}, v \in Def(n) \\ S_B(n, v, B_n) &= \bigcup_{m \in M(n,v,B_n)} S_B(m, v, B_n) \\ M(n, v, B_n) &= \{m \in N(G) \mid m \rightarrow^v n \in E_B(R(B_n))\}, v \in Use(n) \end{aligned} \tag{1}$$

Algorithm 1 represents the computation of the nodes of the block-based slice, which are nodes that may affect the value of a variable v within the region R, given the nodes and edges that belong to it.

Algorithm 1 Block-based Slice

```
1: procedure SliceNodes(node, variable, nodes, edges)
2:     result ← {}
3:     if variable is null || variable ∈ Def(node) then
4:         result ← result ∪ BackwardsTraversal(node, edges)
5:     else if variable ∈ Use(node) then
6:         result ← result ∪ BackwardsTraversal(node, edges)
7:         for each defNode ∈ IncomingDataDependencies(node) do
8:             result ← result ∪ BackwardsTraversal(defNode, edges)
       return result
```

Algorithm 2 represents the implemented backwards traversal algorithm used by Algorithm 1, given the edges that belong to block-based region R. For aesthetics

reasons, Algorithm 2 shortcuts incoming control dependencies and incoming data dependencies to "IncControlDeps" and "IncDataDeps", respectively.

Algorithm 2 Bottom-up Traversal of the PDG

```
1: procedure BackwardsTraversal(initial, edges)
2:     result ← {initial}
3:     Add initial to queue
4:     while queue is not empty do
5:         node ← Dequeue(queue)
6:         for each dep ∈ IncControlDeps(node) ∪ IncDataDeps(node) do
7:             if dep ∈ edges then
8:                 Add Source(dep) to result
9:                 if Source(dep) is not visited then
10:                    Add Source(dep) to queue
       return result
```

As a means to extract the slice to a different action, it is needed to evaluate whether or not the nodes can be removed from the original one. For that, Maruyama defines as indispensable I_B the nodes that belong to the slice S_B but should not be removed from the action to preserve its original behaviour. Thus, indispensable nodes should exist both in the original and the created action [Tsantalis and Chatzigeorgiou, 2011]. This nodes are indispensable due to existing control or data dependencies from slice nodes S_B to non-slice nodes U_B (Equation 2).

$$\begin{aligned} &I_B(S_B, U_B, v, B) = I_{CD} \cup I_{DD}, \\ &I_{CD} = \{q \in N(m) \mid (q \in S_B(p, u, B) \vee q = p) \wedge p \in N_{CD}(S_B, U_B) \wedge u \in Use(p)\}, \\ &I_{DD} = \{q \in N(m) \mid q \in S_B(p, u, B) \wedge p \in N_{DD}(S_B, U_B, v) \wedge u \in Def(p)\} \end{aligned} \tag{2}$$

The computation of the slice nodes with at least one control dependency to non-slice nodes N_{CD} is given by Equation 3.

$$N_{CD}(S_B, U_B) = \{p \in N(m) \mid p \rightarrow_c q \in E(m) \wedge p \in S_B \wedge q \in U_B\} \tag{3}$$

The computation of the slice nodes with at least one data dependency to non-slice nodes N_{DD} is is given by Equation 4.

$$N_{DD}(S_B, U_B, v) = \{p \in N(m) \mid p \rightarrow_d^u q \in E(m) \wedge u \neq v \wedge p \in S_B \wedge q \in U_B\} \tag{4}$$

The aforementioned equations use the following definitions:

N(m) represents the nodes of the PDG.
E(m) represents the dependencies of the PDG.
$\mathbf{S_B}$ represents the slice nodes.
$\mathbf{U_B}$ represents the remaining nodes $U_B = N(m) \backslash S_B$.
Use(p) is the set of variables used by node p.
Def(p) is the set of variables defined by node p.
$\mathbf{N_{CD}}$ is the set of nodes $p \in S_B$ with a control dependency to a node $q \in U_B$.

N_{DD} is the set of nodes $p \in S_B$ with a read after write (RAW) data dependency (due to variable v) to a node $q \in U_B$.
IncomingDataDependencies(p) is the set of RAW data dependencies from node $q \in N(m)$ to node p.
IncomingControlDependencies(p) is the set of control dependencies from node $q \in N(m)$ to node p.

The action in Figure 1 was subject to the implemented algorithm, producing multiple slicing opportunities for each variable within each region. Figure 3 represents one of the slicing opportunities generated by the algorithm. It illustrates the block-based slice for variable "RenterPoints" at the block-based region $R(B_2)$ delimited by the dashed box. Circled in green are the removable nodes and in red are the indispensable nodes that will be duplicated after slicing.

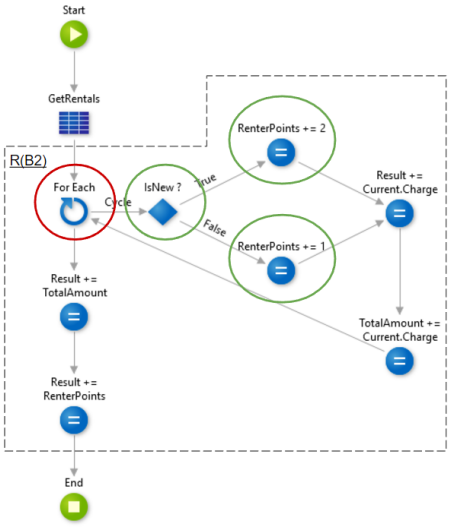


Fig. 3: Block-based slice of "RenterPoints" in $R(B_2)$

However, Maruyama's algorithm does not guarantee behaviour preservation related with the duplication of statements. Tsantalis and Chatzigeorgiou [2011] presented a set of rules regarding behaviour preservation, the most relevant of which for this paper are the preservation of existing anti-dependencies (write after read) and preservation of existing output-dependencies (write after write). These dependencies should be preserved if they start in a node that will exist in the original action after slicing and end in a node that will be removed from the original action after slicing. If not, the order or execution will be changed after extracting the slice [Tsantalis and Chatzigeorgiou, 2011], affecting the action's original behaviour.

5 Case Study

The selected case study is composed by 9 modules developed in the OutSystems platform for some company's purposes. These modules provide employees with means of regression tests analysis, support for new incomers' integration processes, employee information listing, wage premium management and holiday scheduling.

The modules were chosen for their characteristics that are adverse of software quality as they were maintained in rotating team environments. Currently, the modules consist in 1643 actions that contribute, generally, to the system's complexity. The conditions under which the modules were developed and maintained throughout their life cycles are prone to design flaws such as Long Actions.

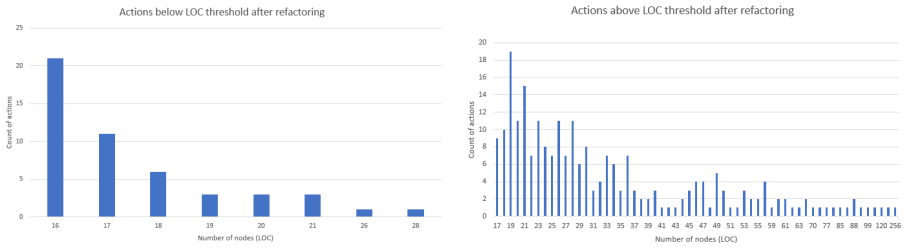
As previously mentioned in Section 4, the selected threshold values are: 15 nodes for LOC and 4 units of CC, i.e., the selected actions for refactoring are the ones whose LOC or CC is higher than the corresponding threshold value.

From the total 1643 identified actions, 331 of them (≈20%) have more than 15 nodes. Around 88% of the actions that are above the LOC threshold (290 actions out of the 331 evaluated actions) produced at least one refactoring opportunity. After refactoring, 49 actions (≈17% of the 290 actions that were above the LOC threshold) reduced their complexity to a point where refactoring is no longer needed, i.e., dropped below the LOC threshold. The remaining 241 actions (≈83% of the 290 actions that were above the LOC threshold) reduced their complexity but are still above the LOC threshold. Table 1 sums up this analysis, given the LOC threshold value at 15 nodes.

Table 1: LOC threshold analysis

	Number of actions	**Percentage (%)**
Total number of actions	659	100%
Number of actions above the LOC threshold	331	20%
Number of actions above the LOC threshold with at least one refactoring opportunity	290	88%
Number of actions below LOC threshold after refactoring	49	17%
Number of actions above LOC threshold after refactoring	241	83%

The 49 actions that once were considered complex but no longer are have a number of nodes mostly ranged between 16 and 20 nodes, as Figure 4a suggests. This may infer that the actions that dropped below the LOC threshold are relatively close to it. The remaining 241 actions that are still complex after refactoring have a number of nodes ranged between 17 and 256 nodes, as Figure 4b suggests. Thus, one can infer that 1 single application of this technique may not be enough. As a matter of fact, 234 actions (≈97% of the actions that are still complex after refactoring) produced at least one more valid slice.



(a) Nodes that dropped below LOC threshold after slicing

(b) Nodes that remained above LOC threshold after slicing

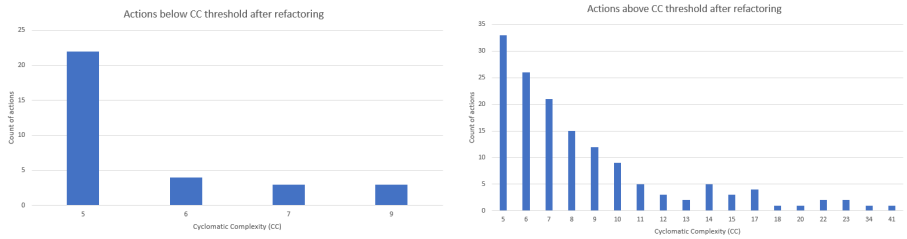
Fig. 4: LOC distribution after slicing

From the total 1643 identified actions, 199 of them (≈12%) have a cyclomatic complexity higher than 15 nodes. Around 89% of the actions that are above the CC threshold (178 actions out of the 199 evaluated actions) produced at least one refactoring opportunity. After refactoring, 32 actions (≈18% of the 178 actions that were above the CC threshold) reduced their complexity to a point where refactoring is no longer needed, i.e., dropped below the CC threshold. The remaining 146 actions (≈82% of the 178 actions that were above the CC threshold) reduced their complexity but are still above the CC threshold. Table 2 sums up this analysis, given the CC threshold value at 4 units.

Table 2: CC threshold analysis

	Number of actions	**Percentage (%)**
Total number of actions	659	100%
Number of actions above the CC threshold	199	12%
Number of actions above the CC threshold with at least one refactoring opportunity	178	89%
Number of actions below CC threshold after refactoring	32	18%
Number of actions above CC threshold after refactoring	146	82%

The 32 actions that once were considered complex but no longer are have a CC value mostly ranged between 5 and 9 units, as Figure 5a suggests. This may infer that the actions that dropped below the CC threshold after slicing are relatively close to it. The remaining 146 actions that are still complex after refactoring have a CC value ranged between 5 and 41 units, as Figure 5b suggests. Thus, one can infer that 1 single application of this technique may not be enough. As a matter of fact, 115 actions (≈79% of the actions that are still complex after refactoring) demonstrated the ability to produce at least one more valid slice.



(a) Nodes that dropped below CC threshold after slicing

(b) Nodes that remained above CC threshold after slicing

Fig. 5: CC distribution after slicing

6 Conclusions and Future Work

The implemented solution aims at identifying Extract Action refactoring opportunities that best reduce the complexity of modules. As such, the block-based slicing technique presented by Maruyama [2001] was used to extract fragments of code that compute a common variable.

In order to accelerate the study of this approach, some limitations are presented as follows: preparation actions are not refactored because some nodes may declare shared variables; Assign nodes with multiple Assignments are split up in multiple Assign nodes with one single Assignment, i.e., the algorithm can generate slices that remove nodes that were initially not visible to the programmer (although one Assign node with multiple Assignments has computational complexity similar to multiple Assign nodes with a single Assignment).

This work supports that low-code platforms still need refactoring primitives. Furthermore, Section 5 (Case Study) proves that the implemented algorithm reduces the complexity of OutSystems modules. Also, the evaluation was based on the analysis of applications whose complexity is similar to ones developed in an Information Technology department and can be considered as good case study examples. Thus, it is emphasized that this study employed on real modules, i.e., endowed with actions with a good diversity of code structure.

Section 5 (Case Study) confirms the need of a tool that automatically identifies improvements in the OutSystems code to maintain the balance and consistency of its projects, by assisting the programmer with valuable refactoring opportunities. Despite that, it is intended that the programmer can qualitatively evaluate the suggestions presented, adding a subjective factor to the analysis.

As shown in Section 5 (Case Study), a single Extract Action application had limited impact on the overall complexity of the modules, as 17% $\sim$ 18% of actions dropped below the complexity threshold. However, after the application of one Extract Action, the vast majority of the refactored actions can still be refactored, which may induce the possibility to incrementally extract different variables. Thus, it is intended that the algorithm suggests multiple and successive refactorings opportunities to take greater advantage of this technique.

The OutSystems platform allows its actions to have multiple output values for a single routine. Thus, it is intended that the algorithm supports multiple variable extraction. This is relevant in situations where there is a high intersection level in slices of different variables. That is, nodes that were once indispensable could be removed from the original action. Besides that, in some situations, it would no longer be necessary to create two subroutines to remove the computation of two variables. Furthermore, this approach leads to having more or even better refactoring opportunities available.

Bibliography

Frances E. Allen. Control flow analysis. *SIGPLAN Not.*, 5(7):1–19, July 1970. ISSN 0362-1340.

Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Commun. ACM*, 36(11):81–94, November 1993. ISSN 0001-0782.

Jean-franc Bergeretti and Bernard A. Carry. Information-flow and data-flow analysis of while-programs. *ACM Transactions on Programming Languages and Systems*, 7:37–61, 1985.

Gerardo Canfora, Aniello Cimitile, and Andrea De Lucia. Conditioned program slicing. *Information and Software Technology*, 40(1112):595 – 607, 1998. ISSN 0950-5849.

Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9(3):319–349, July 1987. ISSN 0164-0925.

Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. 2000.

Computação Distribuída em Redes Formadas por Dispositivos Móveis

Pedro Sanches, António Teófilo, Filipe Cerqueira, João A. Silva, and Hervé Paulino*

NOVA Laboratory for Computer Science and Informatics
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa, 2829-516 Caparica, Portugal
{p.sanches,fa.cerqueira,jaa.silva}@campus.fct.unl.pt
ateofilo@deetc.isel.ipl.pt herve.paulino@fct.unl.pt

Resumo Nos últimos anos temos visto um aumento significativo tanto no número de dispositivos móveis e nas suas capacidades computacionais, de armazenamento e de comunicação, como também no número de aplicações que necessitam de cada vez mais recursos computacionais e de armazenamento. Atualmente, de forma a lidar com esta necessidade crescente de recursos, estas aplicações fazem uso de serviços suportados por infraestruturas *Cloud*. Esta utilização traz alguns problemas: elevados valores de latência, uso considerável de energia e de largura de banda, e ainda a indisponibilidade de infraestruturas de conectividade. Dado este contexto, para algumas aplicações começa a fazer sentido efetuar parte ou toda a computação localmente nos próprios dispositivos móveis. Neste artigo apresentamos OREGANO, uma *framework* para computação distribuída em dispositivos móveis. OREGANO é capaz de processar conjuntos ou fluxos de dados gerados em redes de dispositivos móveis, sem necessitar de uma conexão à Internet. Contrariamente ao atual estado da arte, onde a computação e os dados são enviados para dispositivos móveis "trabalhadores", a nossa *framework* tem como objetivo mover a computação para onde os dados se encontram, reduzindo significativamente a quantidade de dados trocados.

Keywords: Computação distribuída, computação em redes de dispositivos móveis na periferia, processamento de fluxos, Android

1 Introdução

Os dispositivos móveis estão cada vez mais presentes no dia a dia das pessoas, alterando a forma como estas experienciam o seu quotidiano, quer na consulta de emails, no acesso a redes sociais ou mesmo pelo uso de aplicações computacionalmente mais exigentes, como é o caso de aplicações de realidade aumentada.

* Este trabalho foi parcialmente financiado pela FCT-MCTES, no contexto do projeto de investigação CMUP-ERI/FIA/0048/2013, do plano estratégico PEst-UID/CEC/04516/2013 e bolsa de investigação SFRH/BD/99486/2014.

Nos últimos anos temos assistido a um aumento significativo do número de dispositivos móveis e das suas capacidades computacionais, de armazenamento e de comunicação. Segundo a Cisco, o número de dispositivos móveis aumentou de 7.6 mil milhões em 2015 para 8.0 mil milhões em 2016 [2]. De acordo com a mesma fonte, espera-se que o número de dispositivos móveis conectados seja de 11.6 mil milhões, em 2021. Sendo ainda possível constatar que, anualmente, tem sido lançado um novo desenho de CPU com mais poder computacional que o seu predecessor [8].

Este aumento de desempenho está intimamente relacionado com a necessidade de garantir uma melhor experiência de utilização [8]. Cada vez mais, as aplicações requerem o uso de um maior número de recursos e geram grandes quantidades de dados que têm de ser armazenados e potencialmente processados. Atualmente, de forma a combater estas necessidades, as aplicações fazem uso de serviços oferecidos por infraestruturas *Cloud*, seguindo uma política de *pay as you go* [12].

Apesar dos serviços *Cloud* poderem ser uma resposta às necessidades discutidas acima, estes também levantam alguns problemas quando utilizados a partir de dispositivos móveis. O consumo considerável de energia e largura de banda, o acesso limitado ou não existente, e a grande quantidade de dados que são enviados, influenciam negativamente a latência sentida, o que pode ser um problema quando certas aplicações necessitam de oferecer um tempo de resposta praticamente imediato ao utilizador [5].

Tendo em atenção o supracitado, começa a fazer sentido, para algumas aplicações, utilizar outros tipos de modelos que façam uso do poder computacional dos dispositivos móveis e da sua proximidade ao utilizador.

Esta ideia de proximidade está na origem de novos conceitos que têm surgido, como *Edge Computing* [15], onde a computação passa a ser movida para perto ou para onde os dados se encontram. A partir deste conceito, novas noções tiveram origem, como *Fog Computing*, onde um grupo de dispositivos comunicam entre si de forma a processar e armazenar dados sem necessidade de infraestruturas *Cloud* [6]. Também o conceito de *Mobile Edge Cloud* surgiu, onde um grupo de dispositivos móveis forma uma *Cloud* na periferia da rede, que irá processar e armazenar dados sem necessidade de recorrer a qualquer tipo de infraestrutura de conectividade [3].

Com o objetivo de reforçar as motivações para o nosso projeto e o problema que pretendemos endereçar, iremos apresentar um cenário motivacional. Considere um evento social, uma festa, onde um sujeito, a Maria, estará presente. Nesta festa, a Maria irá tirar fotografias com o seu smartphone e irá ser fotografada por outros participantes. No final do evento a Maria deseja obter todas as fotografias onde está presente. Para atingir o seu objetivo, esta irá precisar de descarregar uma aplicação que lhe permite ver e descarregar todas as fotografias tiradas na festa por outros utilizadores, necessitando apenas de enviar parte das fotografias que tirou com o seu smartphone para uma *Cloud*. Com esta aplicação, ela tem ainda a opção de utilizar um mecanismo de reconhecimento facial sobre todas as fotos, de forma a evitar ter de as percorrer uma a uma. Este mecanismo

é oferecido por um serviço *Cloud*, uma vez que consome muita energia e demora muito tempo a executar [17].

Apesar da aparente inexistência de problemas com a utilização desta aplicação, a realidade é algo diferente. O tempo de latência experienciado com a utilização da infraestrutura *Cloud* será significativamente maior, quando comparado com uma solução mais próxima do utilizador. Também a configuração prévia desta *Cloud*, do serviço de reconhecimento facial, e ainda a necessidade de informar todos os participantes da existência deste serviço, poderia ser visto como um problema. Se considerarmos a grande quantidade de dados que têm de ser enviados de um dispositivo móvel para a *Cloud* e descarregados da *Cloud* para um dispositivo móvel, surge-nos um aspeto que pode ostentar um problema, uma vez que o utilizador poderá não ter acesso à Internet através de uma rede Wi-Fi, forçando-o a utilizar tecnologias 3G/4G para o fazer, algo que tem um custo monetário associado. A indisponibilidade de infraestruturas de conectividade, devido à inexistência das mesmas, ao congestionamento e por vezes à sua destruição causada por desastres naturais é certamente um obstáculo à utilização da aplicação descrita no cenário motivacional.

Tendo em atenção os problemas referidos acima, começa a fazer sentido para algumas aplicações, processar os dados que são gerados pelos dispositivos móveis, localmente, sem se recorrer a infraestruturas *Cloud*. Deste grupo de aplicações, as mais comuns seriam aplicações que têm conhecimento do ambiente que as rodeia através de sensores nos próprios dispositivos móveis como também no espaço onde é suposto serem usadas, bem como aplicações que são desenvolvidas com o intuito de serem utilizadas em ambientes sociais. Neste grupo de aplicações, não estão inseridas aplicações que necessitam de uma constante conexão à Internet para ter acesso a dados que de outra forma seriam impossíveis de obter. Da mesma forma, aplicações que lidam com quantidades excessivas de dados e que estão longos períodos de tempo, continuamente a processar esses dados, onde não existem períodos de inatividade, não estão inseridas no grupo alvo de aplicações.

De forma a endereçar os problemas previamente discutidos, neste artigo apresentamos Oregano, uma *framework* para computação distribuída, capaz de processar conjuntos ou fluxos de dados, que são gerados por uma rede formada exclusivamente por dispositivos móveis. Oregano é uma *framework* que se baseia na manipulação de conjuntos de dados, agrupados por *Mobile Dynamic Datasets* (MDDS), conforme o que foi proposto em [13]. Os dados que compõem um MDDS são guardados num sistema de armazenamento e disseminação, ciente de tempo, baseado em *tags*, com mecanismos de publicação/subscrição. Oregano permite o processamento de MDDS definidos a partir das tags associadas aos dados publicados na rede.

O atual estado da arte permite que um dispositivo móvel distribua computações e dados para um conjunto de dispositivos móveis, como se de um *cluster* de nós se tratasse. O nosso sistema diferencia-se conceptualmente do estado da arte, uma vez que a computação é movida para onde os dados se encontram, ao invés de os dados serem enviados para onde se fará a computação.

Neste artigo apresentamos as seguintes contribuições: i) proposta e implementação de um protótipo de uma *framework* para computação distribuída desenvolvida para redes formadas por dispositivos Android, capaz de processar conjuntos e fluxos de dados gerados por esses dispositivos, sem necessidade de infraestruturas de conectividade ou Internet (Secção 2); e ii) avaliação do protótipo implementado (Secção 3).

2 Oregano

Oregano é uma *framework* para computação distribuída capaz de processar conjuntos e fluxos de dados que são gerados por uma rede composta por dispositivos móveis, sem requerer infraestruturas de conectividade. A nossa *framework* permite a computação sobre dados que se encontram distribuídos por um conjunto de dispositivos móveis e que estão organizados por MDDS. A cada um destes MDDS está associada uma *tag*, semelhante às *tags* usadas em redes sociais, e.g. '#Festa'. Através do sistema de armazenamento e disseminação, ciente de tempo, com mecanismos de publicação/subscrição [1], é possível fazer publicações de ficheiros usando *tags* e subscrições dessas *tags*. Um dos aspetos que distingue este sistema de armazenamento é o facto de ser ciente de tempo. Este sistema de armazenamento permite a subscrição definindo um tempo de início e um tempo de fim, especificando assim, que apenas se pretende obter notificações sobre ficheiros que foram publicados entre o intervalo de tempo definido. Isto permite obter dados que foram publicados no passado, desde que exista pelo menos um dispositivo móvel com esses dados na rede.

Com base nesta ideia de *tags*, o nosso sistema oferece APIs para: i) desenvolver e oferecer serviços que processam MDDS; e ii) efetuar um tipo especial de subscrição a uma *tag*, indicando que serviço de computação se pretende, de forma a processar todos os dados que foram ou ainda vão ser publicados, com a determinada *tag*, entre um tempo de início e um tempo de fim à escolha.

2.1 Arquitetura da *Framework*

O Oregano é um sistema simétrico, onde cada dispositivo móvel executa a pilha de software (ilustrada na Figura 1). É importante referir que esta *framework* encontra-se a ser desenvolvida no contexto do projeto *Hyrax* [9], utilizando diferentes componentes/sub-sistemas que foram ou estão a ser desenvolvidos por outras equipas, no mesmo contexto.

Rede. A componente da rede é assegurada pelas duas primeiras camadas, a de Ligação e a de Rede. Estas duas camadas são uma direção de investigação por si só e não estão no âmbito deste artigo. O Oregano utiliza-as como de um serviço se tratasse. Através destas camadas é possível formar uma rede entre múltiplos dispositivos móveis, permitindo a sua comunicação utilizando três tipos diferentes de tecnologias de comunicação: Wi-Fi, Bluetooth e Wi-Fi Direct.

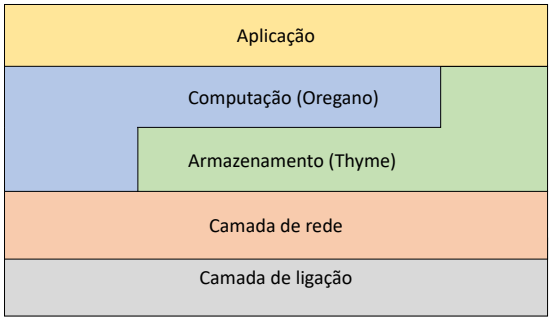


Figura 1: Arquitetura do sistema

Armazenamento (Thyme). Esta camada oferece um sistema de armazenamento distribuído, com mecanismos de publicação/subscrição, capaz de suportar falhas de vários dispositivos móveis, através de estratégias de replicação. Thyme [1] tem por base uma grelha virtual, por onde diversos dispositivos móveis são distribuídos de acordo com a sua localização atual. Este sistema utiliza o conceito de *tags* para operações de publicação e subscrição. Através de funções de dispersão, uma *tag* é mapeada numa das células da grelha, sendo que todos os elementos numa determinada célula são responsáveis por guardar subscrições e notificar utilizadores que novos objetos foram publicados. Nesta camada quando um objeto/ficheiro é publicado este fica associado a pelo menos uma *tag*.

Computação (Oregano). A camada de computação corresponde à *framework* apresentada neste artigo. O Oregano permite o processamento distribuído de dados, tendo como objetivo principal mover a computação para onde os dados se encontram. Oregano vai interagir com a camada de rede, de forma a enviar e receber mensagens pertinentes para a lógica do sistema. Interage ainda com Thyme, dependendo de muitas das suas funcionalidades. Através de Thyme, é possível a replicação de dados, ajudando desta forma a lidar com a entrada e saída de dispositivos móveis da rede, i.e., *churn*, na medida em que existirão mais dispositivos com os mesmos dados, logo esses dados ainda poderão ser processados mesmo que o dispositivo que os publicou tenha abandonado a rede. Acresce que, com base na ideia de os dados se encontrarem presentes em mais do que um dispositivo, poder-se-á dividir algum trabalho, visto que mais dispositivos poderão processar os dados. Similarmente, os metadados guardados por Thyme, são necessários para todo o processo de escalonamento de computação e retorno de resultados.

Aplicação. Como o nome indica, esta camada diz respeito às aplicações implementadas com a *framework* Oregano. A camada de aplicação poderá interagir

com Oregano, usando a API oferecida, podendo ainda interagir com Thyme, de forma a utilizar algumas das suas funcionalidades exclusivas.

2.2 API da *Framework*

A *framework* oferece uma API que pode ser divida em três partes: i) implementação do serviço que irá processar dados, por parte do utilizador da *framework*; ii) gestão do ciclo de vida de um serviço; e iii) utilização de serviços por aplicações, através de uma interface publicador/subscritor com computação associada.

Implementação de Serviços. A implementação de um serviço no Oregano é simples e permite a utilização de qualquer tipo de dados. O utilizador da *framework* tem apenas de implementar dois métodos, sendo um deles utilizado para fazer pré-processamento sobre um objeto, e o outro para o processamento sobre um conjunto de objetos. O utilizador da *framework* terá ainda de garantir que os objetos usados têm métodos para serialização.

O processamento de dados num serviço baseia-se na ideia já abordada na Secção 1, onde múltiplos objetos constituem um MDDS. Tendo em atenção esse aspeto, são oferecidas duas classes, um MDD onde são guardados os dados, e um MDDStream que utiliza Java *streams* [18]. MDDStream oferece uma API bastante semelhante à que é oferecida pelo Java, permitindo a utilização de diversas operações de transformação sobre conjuntos de dados. Por esta razão e pelo facto de conseguir oferecer possíveis melhorias em termos computacionais, comparativamente à utilização de estruturas de dados em memória, Java *streams* foi a escolha indicada para o que pretendemos oferecer com a nossa *framework*.

O método de preProcess recebe como argumentos: i) um objeto do tipo genérico Input, que irá ser pré-processado e o resultado desse pré-processamento persistido de forma a ser utilizado quando process for chamado pela *framework*. ii) uma coleção de objetos do tipo genérico Args, que poderá ser usada pelo utilizador da framework conforme a implementação feita do serviço.

O método de process recebe como argumentos: i) um MDDStream do tipo genérico PreProcessInput, sobre o qual irão ser feitas transformações de forma a retornar os resultados pretendido pelo utilizador da *framework*. ii) uma coleção de objetos do tipo genérico Args, que poderá ser usada pelo utilizador da framework conforme a implementação feita do serviço.

Assinaturas:

- PreProcessedInput preProcess (Input value, List<Args> serviceArgs)
- MDDStream<Output> process (MDDStream<PreProcessedInput> stream, List<Args> serviceArgs)

Gestão do Ciclo de Vida de Serviços. Relativamente à segunda parte da API, são oferecidos três métodos que permitem adicionar, modificar, e remover serviços que são implementados pelo utilizador da *framework*. Nós asssumimos que estes serviços são previamente instalados nos dispositivos dos utilizadores finais da aplicação, para estes poderem processar dados. Desta forma nunca será enviado código para ser executado remotamente.

Assinaturas:
- addService (UUID serviceID, Class<? extends Service> service)
- removeService (UUID serviceID)
- replaceService (UUID serviceID, Class<? extends Service> service)

Utilização de Serviços. Esta parte da API refere-se à utilização de serviços pelas aplicações, através da interface publicador/subscritor com computação associada. Para tal são disponibilizados dois métodos, uma publicação e uma subscrição, sendo estas as operações principais oferecidas pela *framework*. A publicação é especial, uma vez que permite fazer pré-processamento sobre o objeto a publicar. O pré-processamento, é um dos métodos implementados num serviço definido pelo utilizador da *framework*. O método de publicação recebe como argumentos: i) um objeto a ser publicado; ii) uma *tag*, que irá permitir associar o objeto a essa mesma *tag*; iii) uma descrição do objeto, por exemplo um *thumbnail* de uma imagem; iv) um *handler* que irá indicar se o objeto foi publicado com sucesso ou não; v) uma variável booleana que permitirá indicar se se pretende ativar replicação ativa para o objeto a ser publicado; vi) um identificador do serviço a ser utilizado para pré-processar o objeto; e vii) argumentos do serviço, que serão passados ao serviço e utilizados conforme a implementação definida pelo utilizador da *framework*.

A segunda operação oferecida por esta parte da API é um tipo especial de subscrição. Esta é uma subscrição com computação associada, que permite processar um conjunto ou fluxo de dados, retornando depois os seus resultados ao dispositivo que chamou este método. A principal diferença entre o processamento de conjuntos de dados e de fluxo de dados, passa por o último permitir processar objetos à medida que estes vão sendo publicados, ao invés de apenas processar objetos que foram publicados no passado.

O método de subscrição recebe os seguintes argumentos: i) uma *tag*, que indica que todos os objetos publicados com esta *tag* deverão ser processados, mediante algumas condições, como o tempo de início e o de fim; ii) um tempo de início, que indica que apenas os objetos publicados com a *tag* após este tempo serão processados; iii) um tempo de fim, que indica que apenas os objetos publicados com a *tag* antes deste tempo serão processados; iv) um *handler*, que irá retornar resultados, disponibilizar informações relativas ao estado do pedido de computação e permitir controlar o pedido de computação (interrompendo-o ou continuando-o); v) um identificador do serviço a ser utilizado para processar todos os objetos publicados com a *tag* entre o tempo de início e o tempo de fim; e vi) argumentos do serviço, que serão passados ao serviço e utilizados conforme a implementação definida pelo utilizador da *framework*.

Assinaturas:
- publish (Object object, Tag tag, byte[] description, Handler handler, boolean activeRep, UUID serviceID, byte[][] serviceArgs)
- subscribe (Tag inputTag, Time startTime, Time endTime, Handler handler, UUID serviceID, byte[][] serviceArgs)

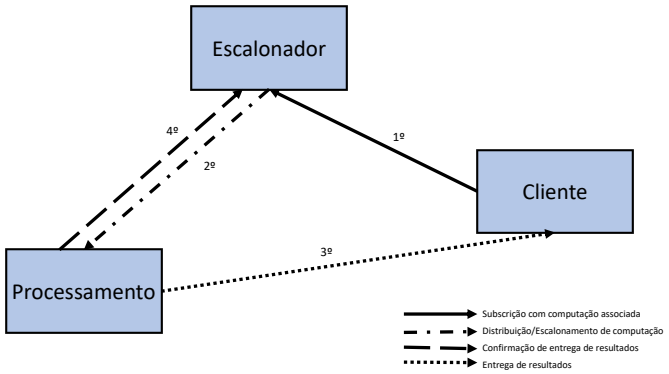


Figura 2: Componentes do sistema

2.3 Componentes

O nosso sistema pode ser dividido em três componentes principais: i) cliente; ii) escalonador; e iii) processamento. Estas três componentes interagem entre si para garantir a conclusão de um pedido de computação com sucesso. É importante referir que os componentes presentes na Figura 2 não pertencem necessariamente a um mesmo dispositivo móvel em simultâneo. Todos os dispositivos móveis do sistema têm estes três componentes, sendo que num exemplo base, cada componente faz parte de um dispositivo móvel diferente.

Cliente. A componente cliente é responsável por permitir à aplicação fazer subscrições, com computação associada, a uma *tag*. Este componente controla as subscrições com computação associada feitas, gerindo a receção dos resultados das mesmas. Um pedido de subscrição com computação desencadeia duas subscrições na camada Thyme: a primeira sobre a *tag* recebida no pedido e a segunda sobre uma *tag* de resultados, que irá oferecer aos dispositivos que irão processar os dados, a possibilidade de publicar esses resultados.

A componente cliente fica responsável por controlar toda a chegada de resultados e oferecer aos utilizadores a possibilidade de *parar* ou *continuar* um pedido de computação. A opção de *parar* encontra-se disponível em qualquer momento, desde o início da submissão do pedido de computação até à chegada de todos os resultados. A opção de *continuar* vai estar dependente de diversos fatores, entre os quais: a quantidade de dados total que já foram processados; a quantidade de dados que ainda não foram processados; se novos elementos foram publicados, apenas no caso de o tempo de expiração da subscrição ser no futuro.

Escalonador. A componente escalonador é responsável por gerir as subscrições com computação associada que recebe, distribuindo diversas computações. Um dispositivo é escolhido como escalonador quando a *tag* passada ao componente cliente, no momento da submissão da subscrição com computação, é mapeada na

célula onde este se encontra, significando que o dispositivo escolhido tem todos os metadados associados à *tag* subscrita. Desta forma, este dispositivo irá ter acesso a diversas informações sobre os objetos a serem processados, nomeadamente as localizações/endereços onde estes se encontram, podendo escolher os dispositivos que irão processar os dados. Esta componente escolhe um x número de objetos a serem processados de uma só vez, forçando a aplicação a escolher se pretende que os próximos x número de objetos sejam processados ou não. Esta abordagem é igual, quer o pedido de computação seja sobre um conjunto de dados (tempo de início e fim são um tempo no passado) ou sobre um fluxo de dados (tempo de fim será um tempo no futuro).

Após o envio de mensagens de computação aos dispositivos que contêm os dados a serem processados, o dispositivo escalonador irá controlar através de um mecanismo de *heartbeat*, se todas as mensagens enviadas foram recebidas e se todos os dispositivos que receberam as mensagens continuam a processar os dados escolhidos sem erros. Caso algum falhe, a componente escalonador é responsável por redistribuir as mensagens de computação falhadas.

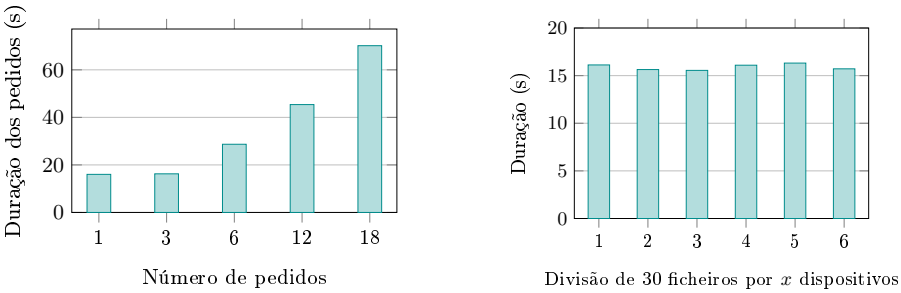
Processamento. A componente processamento é responsável por criar um MDDS com os objetos que se encontram localmente armazenados na camada Thyme. Através do identificador do serviço e dos argumentos do serviço passados na mensagem trocada entre a componente escalonador e este, a componente processamento irá inicializar o serviço, fornecendo-lhe os dados a serem processados e os argumentos do serviço. Após a execução do serviço, esta componente irá decidir entre publicar os resultados com uma *tag* de resultados ou notificar a componente cliente, que irá notificar o utilizador do paradeiro dos dados de forma a que este possa descarregar os resultados para o seu dispositivo.

3 Resultados Experimentais

Foi utilizada uma aplicação muito simples, semelhante à que é descrita na Secção 1, sendo que ao invés de fotografias, são utilizados textos. Esta aplicação em concreto, cria árvores de sufixos a partir dos textos, de forma a encontrar padrões nesses textos, retornando os textos onde são encontrados esses padrões. Isto é comparável a ter-se uma fotografia e retirar certas características dessa fotografia, de forma a que através dessas características fosse possível aplicar um algoritmo de reconhecimento facial.

Os testes foram efetuados utilizando seis dispositivos móveis, onde se foi variando a utilização destes consoante os cenários a serem avaliados. Os três primeiros dispositivos são Motorola Moto G (2nd. gen), Quad-core 1.2 GHz de CPU Cortex-A7, 1 GB RAM, 8 GB de armazenamento, Wi-Fi 802.11 b/g/n, Android Nougat 7.1.2. Os três segundos dispositivos, são Motorola Nexus 6, Quad-core 2.7 GHz, 3 GB RAM, 32 GB de armazenamento, Wi-Fi 802.11 a/b/g/n/ac, Android Nougat 7.0. Todos os testes foram realizados enquanto os dispositivos móveis se encontravam ligados a um hotspot Wi-Fi.

Foram feitos dois testes, tendo o primeiro incidido sobre a duração da conclusão de um conjunto de pedidos de computação (Figura 3a). Sendo que o segundo



(a) Tempo de computação de pedidos.

(b) Tempo de computação mediante a divisão de ficheiros por dispositivo

Figura 3: Resultados experimentais.

pretendeu avaliar o tempo de computação de um mesmo conjunto de ficheiros, mas estando estes ficheiros divididos por diferentes conjuntos de dispositivos móveis (Figura 3b). Nestes testes foram utilizados cinco ficheiros de texto distintos, cada um com uma média de 630 palavras.

No primeiro teste avaliámos cinco cenários, onde em cada um se fez diferentes números de subscrições com computação associada a *tags* que tinham sido utilizadas na publicação de 5 ficheiros. Foi possível observar que nos dois primeiros cenários, em que cada dispositivo móvel publicou um conjunto de dados e submeteu um pedido de computação sobre um conjunto de dados diferentes, que o tempo de computação dos pedidos foi bastante similar. Nos cenários três, quatro e cinco, é possível observar um crescimento do tempo de computação de aproximadamente 75%, 60% e 55%, respetivamente, algo que se deve ao facto de passarmos a ter seis dispositivos, cada um a tratar de um, dois e três pedidos de computação, respetivamente.

No segundo teste avaliámos seis cenários, sendo que em todos os cenários, 30 ficheiros foram publicados com a mesma *tag*. Nos cenários testados, foi-se aumentando gradualmente o número de dispositivos móveis que publicavam ficheiros, escolhendo-se desta forma os dispositivos móveis que processavam dados. Por exemplo, no primeiro cenário, apenas um dispositivo publicou ficheiros, fazendo com que apenas este tivesse todos os dados, como tal a computação é toda feita por ele. No segundo cenário, dois dispositivos publicaram ficheiros, sendo que apenas esses dois puderam processar dados, e assim sucessivamente para todos os cenários. Não nos foi possível concluir como é que a partição dos dados influencia o nosso sistema, algo que se deve muito provavelmente à simplicidade do serviço utilizado para testes.

4 Trabalho Relacionado

Serendipity [14] é um sistema que visa oferecer a dispositivos móveis recursos computacionais remotos, disponibilizados por outros dispositivos móveis na vi-

zinhança. Este sistema é capaz de lidar com a conectividade intermitente dos dispositivos móveis e tem noção do ambiente que o rodeia.

Service-Oriented Heterogenous Resource Sharing (SOHRS) [11] descreve uma arquitetura e uma *framework* matemática para partilha de recursos heterogéneos, através de funções de utilidade orientadas a serviços.

FemtoClouds [7] são *Clouds* formadas por vários dispositivos móveis e baseia-se na ideia de *Cloudlets*. No entanto, a computação/processamento é todo feito por dispositivos móveis, sendo que a distribuição das múltiplas tarefas é assegurada por um dispositivo fixo/dispositivo de controlo.

MClouds [10] são *Clouds* compostas por dispositivos móveis que enviam computações para outros dispositivos móveis ou para uma *Cloud* fixa, quando uma determinada tarefa não consegue ser completada no dispositivo de origem por falta de recursos. Este sistema segue a arquitetura de mestre e escravo.

Honeybee [4] é uma *Cloud* móvel, localizada na periferia, composta por dispositivos móveis que partilham recursos. Neste sistema existe a ideia de um nó mestre que tem uma fila de trabalhos, de onde diversos nós "trabalhadores" irão retirar trabalho.

P3-Mobile [16] é um sistema de computação paralela, baseado num outro sistema paralelo ponto-a-ponto P3, que foi adaptado para funcionar num ambiente móvel. É um sistema que permite a divisão dinâmica de tarefas.

Comparativamente com a nossa *framework*, apenas SOHRS e P3-Mobile suportam diferentes tipos de protocolos de comunicação. A nossa *framework* partilha ainda várias semelhanças relativamente a abordagens de distribuição de computação, com a maioria das outras soluções, na medida em que se segue uma abordagem de atribuir/dar trabalho, ao invés da procura de trabalho, algo que é feito em P3-Mobile e no Honeybee. Também a capacidade de lidar com a entrada e saída de dispositivos na rede, é um aspeto comum a todas as soluções, inclusive a nossa. No entanto existem aspetos que distinguem a nossa solução das outras. A nossa *framework* segue uma arquitetura de par a par, ao invés de mestre escravo. O modelo de programação seguido pela nossa solução também se distingue dos demais, na medida em que é um modelo do tipo de fluxo, contrariamente aos modelos baseados em tarefas. Um dos outros aspetos diferenciadores é a capacidade da nossa *framework* suportar o processamento de conjuntos e fluxos de dados, algo que nenhuma das outras soluções faz, suportando apenas conjuntos de dados.

5 Conclusões

Neste artigo apresentamos Oregano, uma *framework* para computação distribuída em dispositivo móveis Android. Esta *framework* permite processar conjuntos ou fluxos de dados que são gerados por uma rede composta por dispositivos móveis, sem necessitar de uma infraestrutura de conectividade.

Com base nos testes feitos, foi possível observar uma ligeira melhoria na latência da operação de subscrição com computação associada. No entanto, os testes não foram suficientes para se retirar conclusões sobre a *framework*. Reserva-se

UBILocus - Framework for location aware application development

Nuno Sousa, João Nuno Silva
nuno.ribeiro.sousa@tecnico.ulisboa.pt ,
joao.n.silva@inesc-id.pt

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Abstract. Nowadays the number of location-aware applications and services is increasing. The development of these applications relies on the Operating System provided APIs or in libraries provided by specific positioning providers. The use of multiple positioning providers in the same application (for instance GPS and Beacons based indoor positioning) requires the use and integration at the application level of various (and sometimes incompatible) functions and data types. Furthermore, the use of complementary services requires the integration with respect to the coordinate system to use.
In this paper, we propose a middleware that allows the integration and abstraction of multiple positioning providers. We defined the library interface, the application data types, describe its implementation and on usage application. Besides being a positioning provider (using multiple technologies), UBILocus also integrates a route generation module usable by the mobile application.
UBILocus was implemented using the Xamarin Platform, allows the use of two positioning providers (GPS and Aruba Beacons[1]), provides two routing subsystems and seamlessly presents in the application private maps. The implemented UBILocus was tested in an Android application.

1 Introduction

The development of location-aware mobile services requires a myriad of services to calculate the location of the device, provide routes, present maps and provide added value to the user[12]. These usual services are presented in Figure 1.

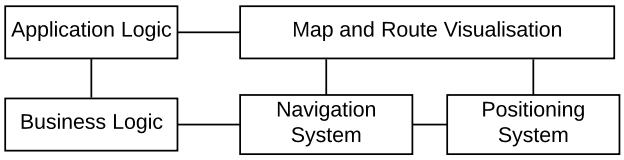


Fig. 1. Services in location-aware application

APIs and libraries to access these services (map visualization, navigation and positioning) are widely available and allow the development of complex systems

and applications to fulfil most stakeholders requirements. These libraries are provided by the Operating System (for instance positioning services based on GPS, Wi-Fi and network cells) or by other companies which are developing additional indoor positioning protocols and services.

The development of location-awareness applications and services are crucial in areas such as tourism[5], smart cities or even retail[4], where accurate position and efficient routing should be provided to the users and services. For instance, in these areas the integration of indoor/outdoor positioning is crucial, but until now the development and integration of various location related services incur problems: i) the application should integrate the various positioning providers APIs and data types, which could be impossible if the number of positioning providers grows; ii) the application should correctly select the positioning service to use at each time; iii) in the case of navigation, the application should be compatible with different coordinates in order to process and correctly display the route; or iv) the coordinates of the points-of-interest (POIs) should also be compatible with the positioning providers. The problem of using multiple location services can also be seen in large campus navigation where user can walk in the outdoor, using the GPS for location, or walk indoor where is necessary to use a more precise and adequate position system [10].

For the mitigation of the presented problems, we propose a middleware that allows the integration of multiple positioning services and routing services that could be accessed by a common API.

UBILocus is a framework that is extensible, allowing the integration of multiple positioning providers, from different vendors that could work in indoor or outdoor environments. It is also a flexible framework, allowing the simultaneous use of those position systems (hybrid solution). The details of those systems (running locally on the mobile device or requiring a remote server) will be hidden from the application developer. Furthermore, the developer will not be aware of the transition of the environment (indoor/outdoor) since the selection of the suitable provider will be handled by UBILocus.

Another functionality of UBILocus is to integrate multiple navigation systems. The access to this component will also be hidden from the developer by means of an API that will allow different routing requests.

This integration will require the definition of a high level location object (usable in the application and compatible with the various components) and the definition of a distributed architecture:

- library linked to the mobile application;
- service running on the mobile device and responsible for forwarding location or routing requests;
- remote server responsible for the integration of the various routing subsystems.

The remainder of the paper is organized as follows. In Section 2, we explain some key concepts regarding navigation systems and positioning systems. In Section 3 we explain the proposed framework. In Section 4 we explain the

implementation details. Finally, we state our conclusions and future work in Section 5.

2 Related Work

The creation of a framework that abstracts the type the positioning and navigation systems involves different research topics. In the present section, we explain the challenges of each of this topics.

2.1 Positioning Systems

A Positioning system is a system that retrieves the position of an object, device or person in a given place. IN this work we divide positioning systems with respect to the type of environment in which will be used:

- **Outdoor Positioning Systems** In Outdoor Positioning Systems are included Systems like the Global Positioning System (GPS)[11], the GLONASS[11] and Galileo [11]. These systems are composed of a constellation of satellites and a receiver. The receiver is able to calculate its own location by measuring the time-of-flight of the signals received from, at least, 3 satellites. After this measurement, the device, based on a trilateration algorithm, calculate its position. To the device operate properly, the satellites need to be in line-of-sight. In urban environment, these systems could obtain an error of 28 meters along a street 95% of the time[14]. This error is due to the line-of-sight requisite not been guaranteed in urban areas.
- **Indoor Positioning Systems** Nowadays, the standard Global Positioning System (GPS) provides a great Outdoor Positioning System, whereas doesn't exist a consensual accepted equivalent for an Indoor Positioning System (IPS). In the past few years all kinds of IPSs[10] appeared based on, for example, Wi-Fi[7], RFID[18], Zigbee, Bluetooth Low Energy(BLE), GSM, Light encoding[8]. More recently are appearing technologies that use a combination of several smartphone sensors, such as, Wi-Fi and Bluetooth[3]. All these indoor positioning technologies, normally, are based on proprietary infrastructure and makes use of custom APIs making the application development harder.

Due to the lack of a Positioning System that covers indoor and outdoor environment simultaneously, UBILocus allow the integration of multiple positioning systems at low cost to developers by providing a common API to all positioning systems. the selection of the most appropriate sub-system is out of the scope of this paper.

2.2 Coordinate Systems

A topic that is tightly coupled with the Positioning System is the Coordinate System. A Coordinate System is essential to uniquely determine the position of

a device. This position can be of three different types[16]: i) **Absolute position** represents a position on the surface of the earth that is characterised by coordinates whose origin is the centre of the earth. The standard coordinate system in use is the World Geodetic System established in 1984 and also known as WGS84; ii) **Relative position** represents a position that is relative to a referential. For example, in the case of the IPSs, normally the user position is given as a position on a given map, *i.e*, this position is relative to the centre of that map. Normally the map is provided as an image thus the coordinate system is centred in the top left of the image and the units of the coordinate system are pixel. Note that a relative position is always represented by a coordinate but this coordinate is relative to a fixed origin (center of the map); and iii) **Symbolic position** represents a location characterised by a name and an identifier. At the application level, this location doesn't need an coordinate associated.

Note that in Outdoor Positioning Systems the most common type of position is Absolute position whereas, in IPSs, the position could be represented in many different ways. For example, in some IPSs with room level precision, *i.e*, only is identified the room where the user is, is common to use Symbolic position that identifies unequivocally the current room. In the Section 3.4 we will explain how this three types of position can be combined to provide an abstract positioning system.

2.3 Mapping services

We define a Mapping Service as a service that provides information about the world. This information should contain a representation of the world or more commonly known as a map. This map helps to provide a contextualized experience providing landmarks (features), paths and a visual representation to the user. In our research, we identified multiple mapping services such as Meridian (used by Aruba Beacons), Google Maps and OpenStreetMaps.

In Table 1 we summarize some properties on the selected mapping services. Note that, the OpenStreetMaps is an open source solution that allows to have

Mapping Services	Supported Environments	Open Source	Public and Documented API	Include Navigation System	World-Wide Coverage	Map Information
Meridian	Indoor and Outdoor	No	No	Yes	No	Private, proprietary data types
OpenStreetMaps	Indoor and Outdoor	Yes	Yes	No	Yes	Public or private, Open Source and easily extended data types
Google Maps	Indoor and Outdoor	No	Yes	Yes	Yes	Public, proprietary data types

Table 1. Comparison between different mapping services

private data. In the case of a navigation application in a museum, this feature allows to maintain the map information only available to the users that access the museum.

2.4 Navigation Systems

A navigation system should provide a route between two or more given locations. Nowadays navigating outside is a relatively common task, where exists services like Google Maps (with Google Directions API) and Bing Maps. Navigating inside, where exits few commercial services, such as, Meridian and indoor.rs, is a relatively new task, although recent observations said that 80% - 90% of the people daily life is spent inside[13]. This fact can be explained by some of the limitations of the IPSs and because, only in the last years, we assist to the generalisation of the use of the smartphones which is used by the majority of the IPSs[6].

During our research, we have identified multiple routing systems but few of them could, out of the box, create a route between a set of locations where some locations could be indoors and others outdoors. The main problem in the identified routing systems is that the provided APIs are not prepared for indoor routing because these systems only works with coordinates that are 2D (a latitude and a longitude). In the indoor case, a coordinate composed only by a latitude and a longitude could identify multiple places because at the same location could exist multiple floors. In Table 2 we summarise some properties on the selected navigation systems. Note that, the GraphHopper routing system uses the OpenStreetMap data. As GraphHoper is Open Source and easily extensible, is possible to make modifications to the routing engine in order to make it compatible with indoor environment.

Navigation Systems	Supported Environments	Open Source	Public and Documented API	World-Wide Coverage
Meridian Routing	Indoor and Outdoor	No	No	No
GraphHopper	Outdoor	Yes	Yes	Yes
Google Directions	Indoor and Outdoor	No	No for indoor case	Yes

Table 2. Comparison between different routing systems

2.5 Existing frameworks

In the bibliography are proposed different frameworks that also try to create a middleware between location-based applications and positioning services. In [17] is presented a framework that integrates multiple positioning systems providing a common interface for accessing the positioning information. This framework also provides a seamless handover between different technologies (indoor and outdoor). Although this framework is easily extensible regarding positioning systems, it can't integrate new mapping technologies or navigation systems. In [19] is proposed a framework for an indoor positioning and navigation system. Although this system can be easily extended with different navigation and positioning systems, it can't handle different types of coordinates system, for instance, Absolute, Relative and Symbolic as defined previously. Midlewhere [15] integrates multiple positioning systems and that can operate with multiple coordinate systems, but is not integrated with a navigation system.

3 UBILocus

From the study of existing platforms, we can conclude that the various systems provide most of the requirements for the application developers, but if it is necessary to integrate various technologies several problems arise, with respect to the development and change of the application, or even with respect to the compatibility of data formats used. The objective of UBILocus is to provide to the developers a framework that hides the implementation and data details of the positioning, routing and mapping services (presented in Figure 1), offering ease of use, interoperability and extensibility.

UBILocus will be called by the application and business logic in order to provide added value to the users/service provider.

3.1 Requirements

In the context of a mobile application, the Positioning component of UBILocus will allow the application, and any existing back-end, to calculate the device location. This location can be used both by the Application or by the Business Logic. The Positioning component should hide the differences between indoor and outdoor environment, as well as, allow the use of multiple and complementary positioning techniques (hybrid positioning). Multiple positioning systems can even be used simultaneously depending on the environment of use.

The main objective of Navigation component is to provide a route between a source location A and a destination location B. This navigation system component should allow the definition of constraints and requirements when calculating the optimal route. For instance, a route should contain intermediate locations, also known as waypoints. Besides allowing the integration of various routing algorithms/subsystems, this component should also be capable of handling the location provided by the various positioning services.

The mapping widgets should also be configurable (to allow changes in implementations) and compatible with the various data types and formats used in the application.

The overall requirements can be summarized as: i) integration and simultaneous use of multiple positioning and routing subsystems; ii) use of private/supplied maps in the mapping subsystem; iii) interoperability between the various positioning, routing and mapping subsystems; iv) Single API;

3.2 Architecture

Figure 2 presents the architecture of the UBILocus. The various components fulfill the various functionality and requirements presented. Although not yet referred, the UBILocus should rely on a server to implement several of its functionality depending on the adopted technologies (positioning and routing) and to host the Business logic of the application (features, logs and routing policies).

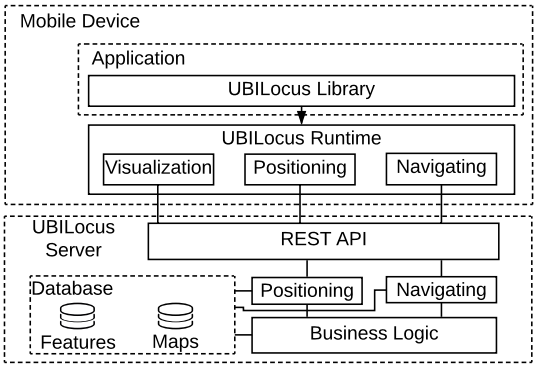


Fig. 2. Architecture of UBILocus.

Besides the library, that exports methods, UBILocus is composed by a run-time on the mobile phone and a server component running co-located with the databases (features and maps) and with the Business logic.

Split between the mobile phone and the server lives the visualization, positioning and routing components.

Although the **Visualisation** component runs on the mobile device to present the user with maps, routes and features, it needs to contact the UBILocus server to fetch the corresponding maps. This maps can be hosted in a private or public service. The limits and the corresponding coordinate system of each map allows the seamless transition between maps as the user moves.

The **Positioning** component will allow the interaction with the installed positioning subsystems (for instance GPS and Beacons). Since a common API is provided (section 3.3), this component is responsible for the selection of the more appropriate positioning algorithm/subsystem and for the transformation of each coordinate system to the one defined in UBILocus (section 3.4). Although GPS works standalone on mobile devices, other more complex positioning systems (for instance based on beacons) can rely on a server component. To fulfill these cases, the UBILocus run-time will also be able to interact with the server to define and calculate the mobile device location.

The UBILocus **Navigation** component is also used to hide and abstract all the complex routing algorithms. This computation is offloaded to the server due to power constrains and necessary information (beacons, available routes, accessibility), the mobile run-time will be mostly responsible for the communication to the server and processing on the responses. Since the routing system should handle indoor and outdoor environments simultaneously, *i.e.*, the selected routing system should handle the different types of locations as described in 3.4.

3.3 Library

The library will hide all the details of the various subsystems providing at the same time, the functionalities required by most of the location-aware applications. The defined functions are presented next.

Positioning System API Taking into consideration the current programming paradigms, the location retrieving will be based on events and callbacks. The API has two methods to configure the location updates: `StartPosListening` (this function receives a parameter corresponding to the minimum time/space interval between location updates), and `StopPosListening`. Each time the location changes (w.r.t. the configuration) an event is launched. This event contains the current location information.

Routing System API The objective of the Routing component is to calculate a route that connects a set of locations. The mobile application does not need to know the algorithm or subsystem and only needs to call the RequestRoute method. This method receives as input a list of locations, and a JSON parameter that defines the constraints (for instance, type of route or information about impaired mobility). The function returns a route that the Business logic best defines.

Map and Route Visualization API Since the application map should display the current user location, routes and features. The API is as follow: `CenterInLocation` (Centers the view of the map in a given location), `DrawFeature` (Receives as parameter a location and additional information and plot the feature in the map), `CleanFeature` (Removes one feature from the map), `DrawRoute` (Receives as parameter a Route as returned by `RequestRoute` and plots it in the map), and `CleanRoute` (Removes the current route).

3.4 Data types

In order to propose a framework and a API that are extensible and allow the interoperability of various subsystems, it is necessary that the used data types follow a common and rich format. The proposed data types for the APIs are as follow and known by each of the UBILocus components. These data types are used to provide a seamless data transfer between components providing a loosely coupled and extensible system.

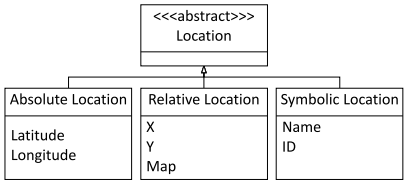


Fig. 3. Location data types

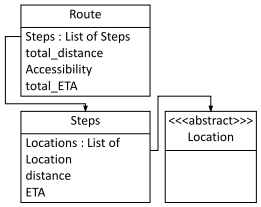


Fig. 4. Route data type

A **Location** is a generic data type that unequivocally identifies a point in space. Since various subsystems use different location representations, we need to define a generic Location type that can be extended to various representations. Figure 3 presents the various sub-classes of locations supported by UBILocus.

Absolute and relative Locations are similar in data but differ in use, since a Relative Location is associated with a Map (for instance building, or floor).

Examples of symbolic Locations are Rooms (with textual identifies) or generic facilities (such as WCs). For the application, these generic facilities do not need a coordinate, but internally will have a actual physical place. Locations represented in different types (e.g. absolute and symbolic) will be matched in the route graphs and displayed in different maps.

Routes (Figure 4) represent the list of Locations that allows a user to go to a set of certain Locations. The way to calculate these routes is hidden from the mobile application developer and the only information presented are the actual points (initial, final and intermediary), the estimated distance and time of arrival.

A **Feature** represents a Point of Interest relevant to the application and is composed of a location, an identifier and additional data.

Since Features, Routes and Locations can be local to an institution, building or private area, **Maps** should also be considered as a UBILocus data type. They will be used in the mobile application to visualize data but will also provide context to the relative Locations. Maps information includes: a Name, a Floor Number, a graphical representation (image) and a unique identifier.

4 Implementation

The first version of the UBILocus library is targeted at the Xamarin Platform. It consists of a set of C# files and binary libraries. This implementation allows the execution of the applications compiled to various mobile platforms (Android and IOS). The run-time is included in the library and linked to the application.

The UBILocus server implements a set of REST APIs (for the various components) and was implemented in Python. This server aggregates the actual implementation/subsystem for the maps storage, routing definition and positioning calculations, as described next.

Positioning System The current version of the positioning component implements two distinct positioning algorithms. This component allows the access to the local GPS service, but also provides access to the indoor Aruba Beacons positioning service. Using these complementary technologies is possible to provide the position of the device indoors and outdoors.

In order to accommodate these two subsystems the positioning component is implemented as described in figure 5. This component has a wrapper that receives requests from the application and redirects them to the suitable subsystem. Since these two systems use different coordinate systems, the positioning component is responsible for the needed standardization.

Current implementation used a simple selection algorithm: if a valid GPS position is calculated with a accuracy of 20 meters or greater, this position is used by the positioning system component. Otherwise, if a valid position is returned by the Aruba Beacons IPS, this position is used.

Routing System The Routing System runs completely on the server (as presented in figure 6), the mobile run-time only forwards requests to the server.

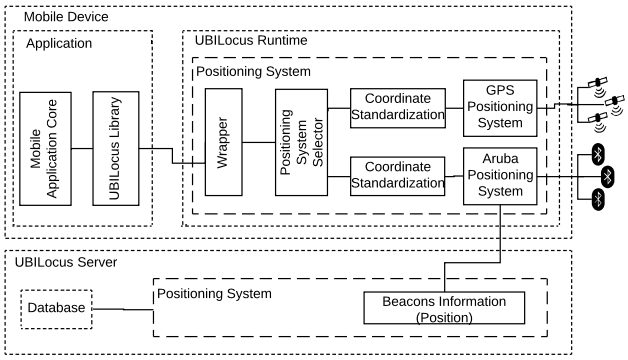


Fig. 5. Positioning System.

The current implementation allows the use of the Meridian[2] or GraphHopper[9] route generator engines, and provide a placeholder for the ordering of locations to visit and engine selection. The main function of the reordering system is to reorder, if necessary, the list of Locations based on a predefined strategy. After the reordering, the list of Locations is passed to the routing engine.

Visualization The implemented Visualisation System uses the open-source OpenLayers JavaScript library (openlayers.org) to display the suitable map: floor plant, features, route and the device position. This JavaScript library is wrapped around a web-view container as presented in Figure 7. This system also performs the download from the server of the suitable map to present in the application

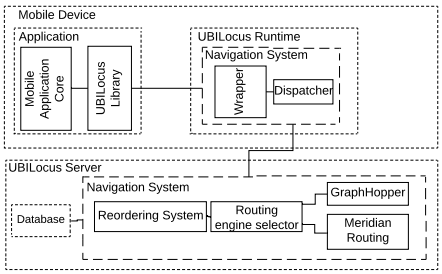


Fig. 6. Routing System

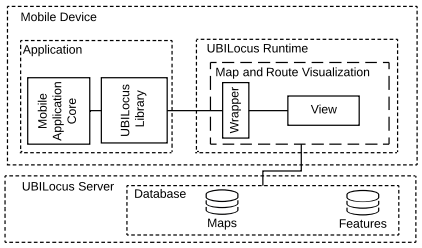


Fig. 7. Map and Route Visualisation.

4.1 Validation

UBILocus was validated with the implementation of the Xamarin ILocate application, allowing users to navigate and get directions in an organization composed of several buidings using indoor (Aruba Beacons) and outdoor (GPS) positioning systems. Figures 8 and 9 present an example of a single route, in yellow, that is composed by a indoor path (left) and outdoor path. Depending on the location UBILocus selected the suitable location mechanism and presented the correct map.

UBILocus also allowed the use of the GraphHopper routing engine with two distinct coordinate systems: GPS (absolute) and Aruba beacons (relative). The GrapHhopper routing engine was extended with the development of a simple "Business Logic" which generate intermediate locations that the user should visit while going to its destination.

The proposed requirements (simultaneous positioning, routing, mapping subsystems and single API) were all accomplished with the design of UBILocus and demonstrated with its implementation and use in the development of mobile application.

5 Conclusions

In this paper, we proposed a middleware, called UBILocus , that allows an easy way to develop location-aware mobile applications that work in indoor and outdoor environments. We provide a single API for accessing the various positioning services. Moreover, UBILocus allows the simultaneous use of multiple positioning subsystems, providing the programmer/user a transparent and seamless transition between different environments.

UBILocus also provides complementary location-aware mobile services such as route generator and map service. Both services could be easily exchanged in order to use the underlying technologies that best fit the requirements and the applicable business logic. All the features provided by UBILocus are hidden and abstracted in order to ease the development.

Although tested on Xamarin/Android the UBILocus system (composed by an API/Library, Runtime and Server), is possible to be used by different programming languages at low cost to the developers. This is possible due to layered architecture, isolation and abstraction, only requiring the porting of a new UBILocus Library, suitable for language/environment.

Also, UBILocus can be extended by adding custom components (that implement additional services), due to a well defined core of data types, and rich API. This extensibility feature was tested with the integration of multiple location/routing services and the demonstrative the business logic component.

As future work, we will continue to integrate new positioning technologies, mainly indoor positioning technologies. Further research has to be done on the

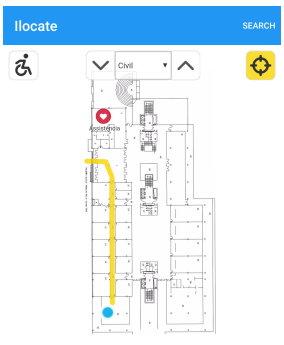


Fig. 8. Building Indoor View.

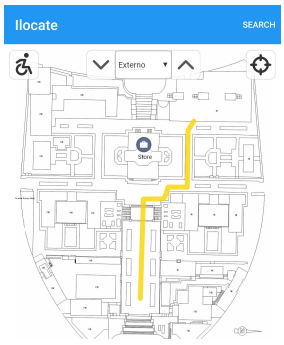


Fig. 9. Campus View.

Estacionamento seguro para bicicletas em meio urbano

Pedro Marcelo Pires Alcobia, João Pedro Barreto
pedro.alcobia@ist.utl.pt, joao.barreto@tecnico.ulisboa.pt

Instituto Superior Técnico, INESC-ID

Resumo Hoje em dia é complicado estacionar a bicicleta na cidade sem nos preocuparmos com a sua segurança. As soluções para aumentar a segurança nos estacionamentos visam o uso de estruturas complexas de arrumo de bicicletas, aumentando assim o custo associado a estes estacionamentos. Em Portugal, os estacionamentos para bicicletas, em geral, são simples e fornecem pouca segurança.
Neste artigo propomos um sistema de guarda de bicicletas com vista a aumentar a segurança das mesmas quando estacionadas, com menor impacto nas infraestruturas existentes. Tal é conseguido através de um sistema informático de monitorização de bicicletas no estacionamento. Estes sistemas não são novidade no mercado [13], porém resumem-se a dois tipos: uns mais simples que usam um pequeno dispositivo(*beacon*) que permite comunicação a curta distância, através de ligações como por exemplo *Bluetooth* ou *RFID*, colocado na bicicleta para alertar outros dispositivos na sua vizinhança; outros mais complexos que permitem a monitorização constante do equipamento, recorrendo a tecnologias de comunicação a longa distância, como por exemplo *GSM*. No primeiro caso existe um problema na falta de disponibilidade do sistema pois é necessário haver um recetor de sinal na proximidade do dispositivo para o localizar. No segundo caso existe o problema do consumo elevado de bateria que é inerente às tecnologias de comunicação a longa distância. Como exemplo temos a aplicação *sherlock* que permite a constante monitorização da bicicleta em qualquer local, no entanto este necessita de mudança de pilha de duas em duas semanas [24].
Desenvolvemos o nosso sistema tendo como base uma solução que permita o baixo custo energético associado ao primeiro tipo de sistemas, no entanto com uma elevada disponibilidade na monitorização de bicicletas, somente onde esta seja necessária. Tal só é possível dada a noção de locais críticos, onde normalmente as bicicletas estão mais desprotegidas. Por exemplo os estacionamentos de bicicletas. Como tal a nossa abordagem envolve a instalação de um pequeno dispositivo recetor no local crítico. Este irá monitorizar as bicicletas na sua vizinhança, permitindo assim uma elevada disponibilidade do sistema no local onde se encontra. Para ser localizada a bicicleta será equipada com um pequeno *beacon* que comunica com o dispositivo recetor através de tecnologias de comunicação de curto alcance, mantendo assim um baixo custo energético associado a este dispositivo. Em caso de roubo a nossa solução notifica imediatamente o dono e potenciais utilizadores considerados de confiança no local. Estes podem ser por exemplo seguranças nos edifícios

perto do local do roubo. Esta abordagem permite uma resposta mais rápida na eventualidade de uma bicicleta estar a ser roubada.
No desenvolvimento desta solução são levantadas algumas questões relevantes ao nível do consumo energético do *beacon* que irá identificar a bicicleta, ou seja de quanto em quanto tempo terá de ser trocada a pilha deste dispositivo. Outro fator a ter em conta será a segurança/privacidade do utilizador ao usar o nosso sistema. A solução proposta neste artigo foca-se nos aspetos da privacidade do utilizador e da disponibilidade do sistema em si, permitindo ao utilizador monitorizar a sua bicicleta quando estacionada.

Keywords: Segurança no estacionamento, bicicletas, Bluetooth, sistemas de localização em tempo real, estacionamento inteligente.

1 Introdução

A bicicleta como meio de transporte é amiga do ambiente, compacta, ágil, fácil de usar, de barata manutenção e saudável [18]. Estas características, aliadas à redução de emissões de CO2, à descongestão de tráfego e à fácil arrumação do próprio veículo, quando comparado com veículos motorizados, tornam este meio de transporte ideal em meios urbanos. Encontrar um espaço seguro onde estacionar a bicicleta é uma tarefa difícil [3], o que leva à falta de adesão a este meio de transporte.

Existem muitos tipos de estacionamentos para bicicletas, vigiados por seguranças [3], vários tipos de bicicletários [2] e até podem ser usadas as infraestruturas da própria cidade para prender a bicicleta, como por exemplo corrimões ou postes.
Estas soluções são, no entanto, insuficientes. Os estacionamentos vigiados ou vedados fornecem bastante segurança; no entanto, por causa do custo da infraestrutura envolvida, é impossível tê-los espalhados pela cidade. Tendo em conta a facilidade com que se consegue desmontar uma bicicleta [1], qualquer ladrão com tempo ou ferramentas suficientes consegue roubá-la [9]. Por isso, os bicicletários e o estacionamento usando as infraestruturas da cidade, apesar de estarem presentes em toda a cidade, não oferecem segurança em todas as circunstâncias aos ciclistas. Durante a noite este problema agrava-se pela falta de pessoas que passam ao pé do local onde a bicicleta se encontra estacionada, dando assim mais tempo ao ladrão para roubar a bicicleta sem ser notado.

O problema da localização de objetos perdidos tem recebido alguma atenção pela comunidade das tecnologias da informação. Como tal, já existem algumas soluções que respondem ao problema do estacionamento seguro de bicicletas, tirando partido das novas tecnologias móveis para permitir a um utilizador monitorizar ou localizar a sua bicicleta em tempo real. No entanto, estas soluções têm limitações: as que permitem uma localização constante gastam muita bateria, necessitando assim de uma fonte de energia contínua, o que em bicicletas é difícil conseguir; as soluções que tiram partido dos meios dos utilizadores, como por exemplo telemóveis, para encontrar os objetos perto de si, não garantem que o objeto seja sempre encontrado.

Neste artigo propomos uma abordagem intermédia que consegue garantir a deteção de bicicletas no bicicletário com reduzido custo energético. Tal é conseguido através dum dispositivo que será colocado no bicicletário com a função de monitorizar a entrada e saída de bicicletas nesse espaço. Este dispositivo, denominado monitor, será o ponto de comunicação entre o dispositivo identificador da bicicleta e o dono da mesma, informando sempre que a bicicleta entre ou saia de um local de estacionamento. Como o monitor se encontra sempre presente no local onde a bicicleta é estacionada, qualquer entrada ou saída da bicicleta será reportada ao dono da mesma, garantindo assim disponibilidade do serviço nos locais mais necessários.
Este monitor poderá ser colocado não só em bicicletários mas também em lojas como cafés ou pastelarias, ou mesmo serviços públicos como polícia ou bombeiros, levando assim o estacionamento seguro de bicicletas a qualquer parte da cidade. Esta rede de locais seguros onde estacionar a bicicleta na cidade irá facilitar também o encontro de bicicletas roubadas.
O monitor tira partido do alcance das comunicações Bluetooth para determinar a presença da bicicleta no estacionamento. Esta abordagem permite simplificar as mensagens enviadas pelo dispositivo na bicicleta, usando o nome do dispositivo Bluetooth associado para transmitir informação ao monitor.

O resto do documento está organizado da seguinte forma. Na secção 2 iremos entrar em mais detalhe sobre as soluções atuais para o problema. Na secção 3 será apresentada a nossa solução, desde o modo de funcionamento como os aspetos de segurança e integridade implementados. Na secção 4 serão apresentados os detalhes da nossa solução, como por exemplo quais as tecnologias usadas, ambientes de desenvolvimento e escolhas de implementação para os vários componentes da nossa solução.

2 Estado Da Arte

Muito trabalho tem sido desenvolvido na área da localização de objetos e utilizadores, seja na localização de objetos perdidos [6, 10, 11], partilha de trajetos diários com amigos [15], ou até usado na monitorização de animais, cadeias de produção e controlo de stock [19]. Certos trabalhos [6, 10, 11] tentam aproveitar os dispositivos móveis dos utilizadores por forma a criar redes de procura de objetos perdidos, alertando sempre que um objeto perdido for encontrado perto de qualquer utilizador. Outros propõem soluções mais robustas usando recetores GPS para localizar o objeto e tecnologias de comunicação a longa distância como GSM para manter uma comunicação permanente entre objeto e dono.

Na literatura, existem três principais tipos de soluções para localizar e monitorizar objetos:

- Globais, que garantem o encontro do objeto em qualquer lado.
- Locais, que garantem o encontro do objeto em certos locais.
  - Com infraestrutura própria para localização de objetos.
  - Sem infraestrutura própria para localização de objetos.

De seguida abordamos cada um dos três tipo de solução em mais detalhe.

2.1 Soluções Globais

As soluções globais permitem uma localização e monitorizar constantemente os objetos em qualquer local. Estas soluções em geral baseiam-se no uso de tecnologia GPS para localizar os objetos e GSM para comunicação à distância. Estas tecnologias consomem bastante energia, necessitando uma fonte de energia contínua [15, 21], o que nalguns casos provoca insatisfação por parte dos utilizadores que vêm a bateria das suas bicicletas electricas a esgotar-se rapidamente [15]. Tal pode ser evidenciado num estudo realizado, onde 34% dos utilizadores queixaram-se de problemas com a bateria [15], como por exemplo, metade da bateria descarregar completamente durante a noite ou que a bateria ficava esgotada muito rapidamente. Também é apresentada nesta secção uma solução que pretende minimizar o impacto na bateria que esses módulos têm. Tal é conseguido dado a presença de outros sensores, giroscópios e acelerómetros, que ditam quando os outros módulos devem entrar em funcionamento [16].

2.2 Soluções Locais com infraestrutura própria

Estas soluções fazem uso de tecnologias como Bluetooth, WiFi e ultra-som ou combinações destas em infraestruturas próprias, com o objetivo de localizar objetos no local onde esta infraestrutura for instalada. As soluções que usam sinais sonoros para localizar os objetos conseguem uma localização muito precisa [22]. Tal deve-se ao conhecido valor da velocidade do som ser baixo, permitindo assim um cálculo mais exato da distância ao objeto, tendo como conhecido o tempo que o sinal demora entre ambos os dispositivos. Outras soluções usam a força de sinal da ligação para medir a distância entre dispositivos [5,12,17,23]. Uma delas [12] combina o uso de pontos de acesso WiFi para encontrar utilizadores dentro do escritório e *beacons Bluetooth Low Energy* (BLE), espalhados pelas divisões do escritório, com o intuito de aumentar a precisão na localização de cada utilizador nas divisões desse escritório. Nesta solução é necessária uma grande constelação de dispositivos Bluetooth e pontos de acesso WiFi para conseguir uma localização com elevada precisão. Estes sistemas contemplam um erro de até 2 metros entre a força de sinal obtida e a distância ao dispositivo [5]. Noutra solução são usados os nomes dos dispositivos Bluetooth [17] para localizar os utilizadores. Esta estratégia permite uma descoberta de utilizadores e das suas localizações sem haver uma ligação Bluetooth estabelecida. Esta troca de informação é feita através dos nomes dos dispositivos Bluetooth. A maior parte destas soluções compreendem a localização de utilizadores, usando para tal os seus telemóveis como dispositivo de identificação. Como tal, poucas soluções requerem pesquisa ou análise sobre os dispositivos usados na localização de objetos [12].

2.3 Soluções Locais sem infraestrutura própria

Em geral, as soluções locais que não requerem infraestruturas adicionais [6,10,11, 20] baseiam-se no uso das tecnologias, Bluetooth ou WiFi, presentes nos dispo-

sitivos do nosso dia-a-dia, como por exemplo telemóveis. Este permite localizar os pr+oprios utilizadores destes dispositivos ou usar os mesmos para localizar objetos à sua volta. Umas soluções, tentam globalizar o encontro de objetos equipados com Bluetooth [6,10,11], ligando todos os utilizadores da mesma aplicação com o intuito de permitir que os objetos perdidos possam ser encontrados quando qualquer utilizador passe perto dos mesmos. Outra solução [20], faz uso das tecnologias de radiofrequência e *Acoustic Ranging* para localizar qualquer dispositivo perto de si. Esta solução foi pensada para ser usada num escritório, onde muitos destes dispositivos se encontram, facilitando o acesso a cada um deles por parte dos utilizadores.

3 Solução proposta

3.1 Arquitetura da solução

A solução apresentada nesta secção tem em conta as observações feitas na secção 2.

A nossa solução pretende dar resposta aos problemas de falta de segurança nos estacionamentos para bicicletas. Como tal, propomos um sistema com infraestrutura própria a ser instalada no local, tal como as soluções apresentadas na secção 2.2. Esta solução envolve colocar um pequeno e barato dispositivo Bluetooth na bicicleta. Este pequeno dispositivo que, por ter um baixo consumo energético, não perturbará o uso diário deste meio de transporte, será usado para identificar a própria bicicleta. Estimamos que uma pilha dure no mínimo 1 ano neste dispositivo.

Dispositivos fixos, que serão colocados em vários locais da cidade, como por exemplo cafés, lojas e bicicletários, irão monitorizar os pequenos dispositivos presentes nas bicicletas. Estes dispositivos fixos, também equipados com tecnologia Bluetooth, têm como objetivo monitorizar a entrada e saída de bicicletas na sua área abrangente. A entrada e saída de novas bicicletas é detetada quando uma bicicleta equipada com um dispositivo móvel se aproxima de um dos dispositivos fixos espalhados pela cidade, criando uma ligação Bluetooth entre ambos que permite informar o dispositivo fixo da presença da bicicleta na sua área. Os dispositivos fixos enviam informação sobre a entrada e saída de bicicletas do seu local abrangente para um servidor central. Este servidor central será responsável por notificar o dono quando a bicicleta for roubada. As notificações serão enviadas não só para o dono, mas também para outros utilizadores considerados seguros, caso existam para esse dado local. Os utilizadores seguros, considerados de confiança, podem ser associados ao local como salvaguarda extra do local, servindo assim de seguranças não oficiais caso haja uma tentativa de roubo. Esta ideia tem como objetivo deixar os ciclistas mais descansados quando estacionam a sua bicicleta num local com utilizadores seguros, pois sabem que alguém estará por perto para ajudar em caso de tentativa de roubo. Tal como quando se pede a um amigo/conhecido para vigiar a nossa bicicleta enquanto estacionada.

Na nossa solução existem três cenários possíveis identificados pelo nosso sistema: o utilizador estaciona a sua bicicleta num local abrangido pelo nosso sistema; o

utilizador sai do estacionamento com a sua bicicleta; a bicicleta sai do estacionamento indevidamente (é roubada). Estes três cenários são identificados pelo servidor central através da informação recebida pelos dispositivos fixos. Em caso de roubo da bicicleta, o servidor central notifica o dono e outros potenciais utilizadores interessados, através da aplicação disponível, o roubo da mesma.
A nossa solução compreende quatro componentes essenciais:

- *Beacon* que comunica a informação da bicicleta
- Dispositivo Geolocalizador envolvido na procura de bicicletas dentro da sua área.
- Servidor Web que processa a informação dos roubos
- Interface com o utilizador

Um Diagrama com o sistema proposto encontra-se apresentado na Figura 1, na qual podemos ver as tecnologias escolhidas na comunicação entre os vários módulos.

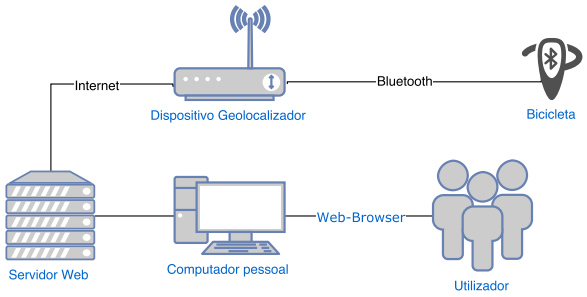


Figura 1. Diagrama do sistema proposto

Componentes da solução. Decidimos usar a tecnologia Bluetooth para nos apoiar na descoberta de bicicletas. Esta escolha tem a vantagem do baixo custo energético inerente a esta tecnologia [8, 14]. Outra característica que nos pode ser util é a descoberta automática de dispositivos à sua volta. Tendo esta característica em conta podemos usar o nome do dispositivo para comunicar a informação necessária [17], que no nosso caso será a identificação da bicicleta no nosso sistema e o seu estado. O estado da bicicleta será qualquer outra informação no *beacon* da bicicleta que nos possa ajudar a identificar um roubo. Na nossa solução usamos um sensor de movimento para detetar movimento da bicicleta que possa identificar uma tentativa de roubo.
Como base para o dispositivo de descoberta das bicicletas foi escolhido um dispositivo fixo. Esta abordagem, dada a presença constante do dispositivo no local, permite a disponibilidade do sistema mesmo durante a noite. O que poderia não ser possível caso fosse usada uma solução móvel para este dispositivo. Esta solução necessita o fornecimento de uma ligação à *Internet* para comunicar com um

servidor central.

Casos de Uso. Nesta secção apresentamos os dois casos com mais impacto para o nosso sistema, a entrada e saída de bicicletas do estacionamento.

Bicicleta entra no estacionamento. No caso de uma bicicleta entrar num local de estacionamento, o sistema procede da seguinte maneira:

- O dispositivo fixo de pesquisa faz a descoberta dos dispositivos à sua volta.
- Ao descobrir uma nova bicicleta envia essa informação para o Servidor Web.
- Caso essa entrada seja de uma bicicleta roubada, então o utilizador será notificado da nova localização da bicicleta.
- O servidor Web guarda a informação destas chegadas, caso seja necessário enviar para a polícia mais tarde.

Este envio de mensagens possibilita o controlo e potencial gestão das bicicletas no estacionamento, levando ao encontro de bicicletas roubadas, melhor gestão de lugares disponíveis e maior segurança.

Bicicleta sai do estacionamento. No caso de saída da bicicleta, o dispositivo fixo só precisa de detetar que um dispositivo que encontrou, na última descoberta de dispositivos na rede Bluetooth, já não se encontra disponível. Quando tal acontecer procede da seguinte maneira:

- O dispositivo fixo de pesquisa envia a informação do dispositivo que não foi encontrado para o servidor Web.
- O servidor Web regista a altura da ocorrência e, caso seja determinado um roubo, envia uma notificação ao utilizador com esta informação.

Para ser detetado um roubo, é necessário conseguir distinguir quem está a sair do estacionamento com a bicicleta, se é o dono ou outra pessoa. Neste momento temos uma solução que envolve ação do utilizador, levando-o a indicar na aplicação sempre que pretender sair com a bicicleta do estacionamento e a que horas o pretende fazer. Caso o utilizador esteja ao pé da bicicleta e queira sair com esta também terá a opção de destrancar a mesma na interface disponível. Estas soluções permitem ao utilizador definir se e quando vai sair com a bicicleta, no entanto requer uma participação ativa por parte do mesmo. Outra solução envolve perceber se, quando a bicicleta saiu do estacionamento, o utilizador estava presente no local. Tal será possível se o utilizador tiver um telemóvel com ligação Bluetooth e o identificador Bluetooth do telemóvel for associado também à bicicleta. Esta solução permite ao utilizador não se preocupar sobre quando irá sair com a bicicleta do estacionamento, no entanto, estes identificadores poderiam ser copiados o que tornaria nula a segurança do sistema.
Como podem haver alguns problemas na retirada de bicicletas do parque de estacionamento, por exemplo não avisar a aplicação que a bicicleta irá sair do estacionamento, iremos dar ao utilizador sempre a palavra final. Ou seja, em caso de roubo, ao ser notificado, o utilizador pode sempre descartar estas notificações de roubo e dizer que foi um mero erro.

Segurança. Com esta grande quantidade de dispositivos móveis e ligações pouco seguras, existem certos aspetos de segurança e privacidade que têm de ser tidos em conta. O principal problema prende-se com a natureza das ligações sem fios, nas quais baseamos a comunicação entre os dispositivos, em particular a ligação Bluetooth entre o dispositivo fixo e o *beacon* presente na bicicleta. Qualquer uma das mensagens enviadas nestas ligações está vulnerável a ataques de repetição de mensagens ou escuta na rede.
Entre o *beacon* presente na bicicleta e o dispositivo fixo de pesquisa, é apenas trocada a mensagem de presença da bicicleta através do nome do dispositivo Bluetooth na rede. Dada a grande quantidade de dispositivos que permitem comunicação através da tecnologia Bluetooth, esta informação pode ser vista por qualquer outro utilizador causando assim problemas ao nível da privacidade dos utilizadores. Com o intuito de resolver este problema de privacidade, o conteúdo irá cifrado com um *nonce*, valor aleatório que será enviado dentro da mensagem cifrada, para não permitir a identificação da presença regular de um dispositivo na área.
Outro aspeto importante sobre a segurança é o local onde fica o dispositivo identificador na bicicleta. Se este não estiver bem preso à bicicleta, pode ser removido e assim eliminando a segurança do sistema. Como tal existe um mecanismo que deteta se este dispositivo foi removido da bicicleta ou não, interrompendo a comunicação deste dispositivo caso tal aconteça. Ao deixar de comunicar este é dado roubado. O último aspeto de segurança a mencionar prende-se com a deteção de um roubo ainda antes da bicicleta sair do bicicletário. Para roubar uma bicicleta presa num bicicletário um ladrão tem de quebrar o cadeado, o que envolve normalmente mexer bastante na bicicleta. Para conseguir detetar este tipo de comportamento, o *beacon* terá um sensor de movimento. A informação sobre a movimentação da bicicleta será enviada por Bluetooth para o dispositivo localizador que por sua vez irá inferir se a bicicleta está a ser roubada ou não.

Interface. A interface vai permitir a comunicação entre o sistema e o utilizador. Esta comunicação, tal como explicada nas secções anteriores, será feita tendo em conta os dispositivos do próprio utilizador e só requer que o dispositivo que corre a interface esteja equipado com uma ligação à Internet. Este site interage com a api do *Google Maps* para fornecer uma representação visual dos locais de estacionamento e das bicicletas.
O utilizador poderá fazer as seguintes operações:

- adicionar ou remover os seus dispositivos (bicicletas).
- adicionar ou remover locais.
- procurar locais.
- localizar as suas bicicletas.
- ver alertas.

4 Implementação

Nesta secção iremos entrar em mais detalhe na implementação do protótipo, explicando como foi desenvolvido cada componente referido na secção 3.1.

4.1 *Beacon.*

Para desenvolver o dispositivo móvel(*Beacon*) presente nas bicicletas, com intuito de as identificar, escolhemos o *CC1350 SimpleLink da Texas Instruments*, pois permite uma solução com o menor custo energético, ligação Bluetooth e módulo de cifra AES incorporados.
Foi usado o *Launchpad* como ambiente de desenvolvimento para o micro-controlador *CC1350* e o Code composer Studio para desenvolver o código. Partimos de uma solução presente nos exemplos deste ambiente de desenvolvimento para comunicação *Bluetooth*. Nesta solução é enviado um sinal de descoberta a cada 160ms. A esta solução foi adicionada a funcionalidade de cifra do nome Bluetooth do dispositivo. Nesta solução usamos uma cifra AES em modo CBC. Esta permite, recorrendo ao uso de *nonce* no inicio do texto a cifrar, mascarar melhor a informação enviada pelo *beacon*.

4.2 Dispositivo Localizador fixo.

Para dispositivo fixo de descoberta de bicicletas numa certa área, escolhemos o *raspberry pi model 3 B*. Este foi escolhido pela sua versatilidade, pois permite um nível adequado de computação, fácil comunicação entre os vários componentes da solução e uma presença constante no estacionamento.
Para detetar os dispositivos *bluetooth* à sua volta foi instalada a biblioteca *Bluez* [7]. Esta biblioteca permite acesso ao modulo *bluetooth* do dispositivo, permitindo a deteção de outros dispositivos à sua volta. O código para detetar dispositivos foi desenvolvido em *python*, dada a facilidade de programação deste tipo de funcionalidades. O serviço a correr neste dispositivo que comunica com o servidor foi programado em Java e comunica com o servidor web através de sockets. Este componente procura dispositivos á sua volta a cada 10 segundos. A informação recebida por estes dispositivos é enviada para o servidor web, caso envolva a entrada ou saída de uma bicicleta.

4.3 Servidor Web.

O servidor web recebe pedidos através de uma interface usando sockets em Java. Este módulo processa a informação enviada por cada localização e tenta perceber se um roubo está a acontecer ou não. Tal é percetível caso o dispositivo localizador tenha enviado uma mensagem de saída de bicicleta e não seja detetada nenhuma intenção de saída por parte do utilizador. Outro caso importante é detetar, através da informação enviada pelo dispositivo localizador fixo, movimento que possa indicar um potencial roubo. Uma bicicleta quando estacionada

encontra-se parada no local, ou seja sem movimento. Ao receber a informação de movimento numa bicicleta estacionada o nosso sistema lança um alerta de potencial intenção de roubo da bicicleta. O servidor espera por três sinais consecutivos de movimento ilícito da bicicleta no estacionamento para enviar o alerta, permitindo que pequenos movimentos na bicicleta sejam ignorados. Esta informação é passada para a interface através de uma base de dados em SQL.

4.4 Interface.

Para interface com o utilizador escolhemos desenvolver um *Web-site*. Este apresenta um ambiente muito permissivo e, apesar de não ter acesso a todos os periféricos do dispositivo por razões de segurança, pode ser executado em qualquer *browser* desde que haja ligação à Internet. Este ambiente não tem uma funcionalidade de notificações predefinida, no entanto é possível realizar essa funcionalidade, tal como demonstrado em [4].
Esta interface foi desenvolvida em JSP. O nosso sistema suporta notificações *Push*, nestas o servidor envia para o *browser* uma notificação sempre que uma bicicleta for roubada. Estas foram escolhidas em pois, comparativamente às notificações *Pull*, não requerem um pedido explicito do utilizador para receber notificações. Esta simplificação permite reduzir o tráfego na rede e o número de pedidos ao servidor. Para implementar esta funcionalidade usamos a noção de *server-sent events*, uma funcionalidade presente no HTML5. A informação sobre as notificações é retirada de uma base de dados SQL com a qual o servidor web comunica. Enviado a informação necessária para o cliente.
Como alguma da informação do nosso sistema pode ser identificada com coordenadas GPS decidimos usar a API do *Google Maps* para nos ajudar nesta representação dos locais de estacionamento e nos locais onde se encontram as bicicletas do utilizador. Esta informação ficará visível num mapa mundo, permitindo um fácil encontro dos locais protegidos.

5 Conclusão e trabalho futuro

Neste documento foi apresentada uma solução com vista a aumentar o nível de segurança dos estacionamentos para bicicletas, esta solução tem em conta várias soluções propostas para localização de objetos.
Também fizemos distinção e comparação entre uma descoberta local ou global dessas bicicletas. Nesta solução escolhemos como base a descoberta local de bicicletas, colocando o foco na rapidez e robustez do sistema em locais predefinidos. Nesta solução apresentámos uma noção de envolver mais a comunidade na prevenção de roubo de bicicletas na cidade, permitindo que vários utilizadores, considerados seguros, possam receber notificações de roubo de bicicleta nos locais onde costumam frequentar. Apresentámos também uma ideia falada por outros autores [15] que visa usar o nome do dispositivo Bluetooth para troca de informação entre dois dispositivos e dos problemas de segurança associados. Este aplica-se à nossa solução pois permite menor consumo de energia, aumentando o

tempo de vida do dispositivo e facilitando a troca de mensagens. No entanto esta solução coloca alguns problemas ao nível de segurança que são falados na secção 3.1. Nessa secção são apresentadas algumas ideias para melhorar a segurança do sistema, entre as quais o uso de chaves criptográficas para preservar a segurança da ligação.
Em termos da interface com o utilizador, tivemos em conta só as que pudessem estar sempre disponíveis. Exemplos destas interfaces seriam as disponíveis em dispositivos móveis ou *Web-sites*. Decidimos escolher os *Web-sites*, pois fornecem a funcionalidade desejada e também porque os dispositivos móveis requerem programação em vários ambientes como *IOS*, *Android* e *Windows Phone*.
No seguimento deste artigo, será feita a avaliação do sistema. Esta consiste em duas fases. Na primeira fase realizaremos testes ao sistema para verificar o consumo energético do *beacon* e o seu alcance. Também será realizada uma avaliação sobre as garantias de segurança e privacidade que o nosso sistema consegue garantir. Numa segunda fase serão realizados testes com utilizadores reais para comprovar a utilidade do sistema no mundo real.

6 Agradecimentos

Este trabalho foi apoiado pelo projeto TRACE (co-financiado pela Comissão Europeia através do contrato nº635266) e por fundos nacionais através da Fundação para a ciência e Tecnologia (FCT) com referência UID/CEC/50021/2013.

Referências

1. Como desmontar uma bicicleta de trilha (tal como acedido a 6/1/2017). `http://pt.wikihow.com/Desmontar-Uma-Bicicleta-de-Trilha`.
2. Guia para construção de bicicletários adequados (tal como acedido a 6/1/2017). `http://www.acbc.com.br/mobilidade/guia-bicicletario/`.
3. Parking your bicycle in amsterdam (tal como acedido a 6/1/2017). `http://www.holland-cycling.com/amsterdam/getting-around-amsterdam/parking-your-bicycle-amsterdam`.
4. Web site notification solution (tal como acedido a 6/1/2017). `http://stackoverflow.com/questions/16860106/real-time-notification-from-web-server-to-client-bowers`.
5. *BluEyes – Bluetooth Localization and Tracking*, 2008.
6. *GPS and Bluetooth Based Object TRacking System*, volume 5. IJARCCE, January 2016.
7. Bluez. Official linux bluetooth protocol stack(tal como acedido a 27/6/2017). `http://www.bluez.org/`.
8. Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.
9. Holland Cycling (Tal como acedido a 6/1/2017). Bicycle theft - how to avoid the pitfalls? `http://www.holland-cycling.com/tips-and-info/safety/bicycle-theft`.

Can Graphs Solve the Geo-aware State Deployment Problem?

Diogo Lima[1,3], Hugo Miranda[1], François Taïani[2]

[1] LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal dlima@lasige.di.fc.ul.pt

[2]Université de Rennes 1, IRISA, ESIR, Rennes, France hamiranda@ciencias.ulisboa.pt

[3]Escola Superior de Hotelaria e Turismo do Estoril, Portugal francois.taiani@irisa.fr

Abstract. A challenge for large scale distributed mobile applications, such as augmented reality games and social networks, is the management of application state. Because the infrastructure is concentrated in datacenters geographically distant from the end users, state flowing between the large number of users and the supporting infrastructure suffers from latency and network congestion degrading user quality of experience.
To mitigate this problem, *Fog Computing* proposes a physical approximation of the servers to the end users, avoiding the use of the Internet backbone. However, gains depend of the system capability to correctly deploy each state component at its most convenient location. Geo-aware state deployment is challenging as it requires a continuous observation and interpretation of the state utilization patterns, expected to evolve with time as new trends and user behavior emerge. This paper proposes a new object graph-based approach for the geo-aware state deployment problem and compares its performance with approaches based on linear algorithms.

1 Introduction

Large scale distributed mobile applications such as smart cities, augmented reality games (e.g. Ingress, Pokemon Go) and social networks (e.g. Foursquare) concurrently connect a large number of participants. Although with distinct degrees of involvement, all manipulate and exchange significant amounts of application state, mostly mediated by the application's supporting infrastructure. The current trend tends to concentrate this infrastructure in Cloud datacenters, implementing a geographical barrier between the end users and the managers of the application state. The latency and jitter that unavoidably results from this distance negatively impacts the application performance.

The Fog Computing approach [1] proposes an approximation of the servers to the end users by deploying surrogates at the network edge, in particular, on access networks. Although this approach has the potential to mitigate jitter and latency and improve performance, it depends on the ability to correctly deploy each software and state component at its most convenient location.

In the particular case of partial application state, a good geo-aware deployment depends of a successful identification of the state utilisation patterns. This is challenging, knowing that each part of the state is expected to exhibit a distinct pattern. As an example, consider a smart-city traveller support service. The predicted time of arrival of each bus is information more likely to be accessed by commuters waiting at the particular bus stop. However, the number of tickets available in the commuter's monthly pass is state expected to follow the commuter. Finally, the number of passengers waiting to travel on one specific bus route is state best managed at some location close to the bus company facilities. The problem is amplified by the consistency requirements that can be found between elements whose ideal location is distinct. A good example would be the average speed and number of passengers on all the buses in a single route.

An efficient implementation of geo-aware state deployment must implement policies that adapt to the observed utilisation patterns considering the most typical location but also the cost of guaranteeing consistency. In addition, the policies must consider the latency induced by the relocation of any component, which will necessarily disturb ongoing distributed transactions. Our problem is in contrast with existing caching solutions, whose main focus is to accelerate the access to replicas of shared data with a read-only status.

This paper approaches the problem of geo-aware state deployment using graphs partitioning algorithms. Originally proposed to partition database items across storage nodes [2], graph partitions are an interesting approach to distribute state components across different locations so that most of the transactions only need to access one. Unfortunately, the original approach does not consider geographical distribution and the impact of state components relocation. To address this limitation, the paper proposes a new object graph-based approach and compares its performance with others inspired on linear algorithms. The goal is to understand if graph partitioning can be used to identify correlations and to evaluate their suitability for geo-aware state deployment frameworks.

2 Problem Statement

The problem addressed in this paper assumes a large scale distributed application where components of the state can be correlated with geographical locations or with mobile users. In line with the Fog Computing model, we assume that our distributed application is supported by hardware located on a datacenter and a number of surrogates: servers located at the edge of the network. Surrogates provide storage and computing power, are connected to each other and to the cloud datacenter through a high speed wired network.

End users devices include desktops, mobile devices and sensors using high speed wired or wireless networks. It is expected that surrogates perform most of the computational effort, in particular, by application clients in their vicinity. Expectations are that most of the traffic produced by clients is handled at surrogates and can be replied without forwarding requests to the datacenter.

To address these expectations, the system must implement a geo-aware state deployment policy, allowing surrogates to store state at its location of interest.

In this model, the geo-aware deployment is managed by a service that decides in run-time the most suitable location of each state component and coordinates its transfer from one location to another. Conceptually, this service acts as an oracle, becoming eventually aware of all state updates and the location of the requester. Awareness is implemented in practice by having surrogates and datacenter servers to forward in the background the logs of operations they perform.

To tolerate faults, surrogates form partial replication groups that replicate their content. To reduce latency and facilitate consistency operations, partial replication groups membership follows a proximity criteria. For simplicity, our model equally considers an "umbrella" global replication group that includes all the surrogates and the datacenter. This global replication group is expected to keep state components widely accessed.

We assume that application state is composed of data that can be either unique to each user or device (personal data), collaborative data relevant to a specific geographical location (geo-aware data), or general application logic data (global data). Depending on the number of partial replication groups involved, state updates can be either *local* or *global*. Local operations are those that access data enclosed on a single partial replication group, while global operations are those where the data accessed is stored in multiple partial replication groups. It is assumed that the surrogate hosting the client requesting the state update participates in the operation.

The complexity involved in the coordination of global operations motivates the oracle to keep commonly accessed state components in the same partial replication group. Each migration decision must consider the cost of the migration itself, the benefits of increasing the number of local transactions and the cost of the global transactions to occur in the future.

3 Related Work

Devising efficient geo-aware deployment algorithms can leverage on previous research results on database partitioning and geo-replicated cloud storage. It should be noted that the problem goes beyond what is tipically addressed by Content Delivery Networks (CDN) like Akamai [1], given that clients of our target applications are expected to manipulate state.

Data partitioning was originally proposed as a response to the scalability and performance needs of database management systems. Its objective is to make a database more manageable by partitioning data into different nodes. Horizontal and vertical partitioning were the first policies to emerge. In horizontal partitioning, the rows of each table are evenly distributed by the partitions. Vertical partitioning follows the same approach but distributing columns by partitions. The rationale behind these approaches is to make the amount of data more manageable while making respectively the full set of attributes of the same rows or

[1] https://www.akamai.com/

the same attributes of all rows in each group. Other mechanisms, sharing partition data across shared-nothing servers have been proposed for example in [3–5]. These share our goal of minimizing the number of distributed transactions. However, they assume that all partitions are geographically collocated, thus ignoring the negative impact of distance on latency.

In contrast with the geo-oblivious approaches pursued by the databases community, cloud storages bring the notion of physical geography to data partitioning and proposed solutions at different scales. At a server-level scale, geo-replicated cloud storages aim to reduce latency and improve load balancing. At a datacenter scale, the goal is to provide fault-tolerance for datacenter level outages. AdaptCache [6] is a server-scale approach that proposes a cooperative and integrated cache framework for web enterprise systems where application servers cooperatively share their caches. Like in our approach, AdaptCache's "oracle" dynamically evaluates and manages object placement. However, it goes a step further and distributes requests across servers to achieve load balancing and facilitate consistency management. This approach is facilitated by the collocation of servers which, in contrast with our system model, permits AdaptCache to ignore the location of the clients. SPANStore [7] is a geo-replicated key-value store that unifies storage available in multiple datacenters into a single framework. The goal is to take advantage of pricing discrepancies to reduce the overall operation cost. This is in contrast with our model which assumes a dynamic setting where the "cost" of each object is represented directly by latency and indirectly by the distance between the client and the server storing the object.

Resilience to datacenter-level outages is addressed for example using distributed transactional SQL databases like CockroachDB [2]. In CockroachDB replica location is decided automatically, based on user configured criteria such as the types of failures to tolerate and the distinct locations made available by the user. Unfortunately, CockroachDB cannot encompass any concerns related with client access latency neither the dynamism associated to mobile applications.

Regardless of the data partitioning policy used, transactions that involve multiple partitions are by themselves a complex subject that deserves some attention. These tend to negatively impact system performance and several solutions try to mitigate their effect in the system. The approach in DS-SMR [8] evaluates the usage patterns to shape the data mapping. Overall, the goal is to design a dynamic and scalable state machine replication to exploit workload locality. Like in our approach, data partitioning is managed by an oracle. However, state reconfiguration follows an oversimplified rule that trades-off latency by the avoidance of distributed transactions and which consists in transferring the data to a single partition prior to initiating the global transaction. Workload analysis and location manipulation can also be found in the geo-replicated storage system SDUR presented in [9]. This paper shows that performance can be improved by giving priority to transactions using a single partition.

[2] https://www.cockroachlabs.com/

4 Geo-aware State Deployment Strategies

Anticipating the future utilization of state items is challenging and can only be efficiently determined if their utilization follows some pattern. Recall from Sec. 2 that in our system model the mapping of state items in surrogates is managed by a dynamic service implemented as an oracle. The oracle collects transaction logs from the surrogates to evaluate the most suitable one to host each state item. This paper reports on the results obtained with graph partitioning algorithms to be executed at the oracle. The goal is to understand the application and environment characteristics where these algorithms outperform other approaches, hereafter named linear algorithms.

Overall, the performance of the algorithms is evaluated considering two metrics. A perfect algorithm would minimize the number of state item transfers between surrogates and maximize the number of transactions that can be performed locally, i.e. using a single surrogate. The following sections present and discuss the algorithms according to their expected performance.

4.1 Linear Algorithms

Linear algorithm evaluate state items independently, i.e. decisions of where each state item must be placed are taken disregarding their participation on transactions with other state items. This paper evaluates the following linear algorithms:

Static (Static). In the Static algorithm, each object is assigned to the surrogate where it was first used and never changes. Considering the performance metrics above, Static trivially minimizes the number of surrogate state items migration at the cost of penalizing latency during distributed transactions. Therefore, it can be considered suitable for scenarios where the cost of distributed transactions is minimal in comparison with the cost of migrating state items.

Move to Next (MtN). At the beginning of each transaction, all state items involved are transferred to the surrogate that initiated the transaction. Items remain hosted on that surrogate until the algorithm is reapplied. MtN is a direct application of DS-SMR [8] and minimizes the number of distributed transactions at the cost of an expectedly large number of item migrations among surrogates.

Aggressive Polling (AP). Items are moved to the surrogate where they have been more used. It is assumed that the algorithm has an infinite memory, what can represent a penalty for the quick reaction to usage pattern changes for long standing state items. However, it is expected to present a suitable balance between the two radical approaches above avoiding spurious state item transfers.

4.2 Graph-partitioning Algorithms

Graph partitioning algorithms aim to improve performance by considering the association between state items in transactions. This is expected to favour the

observation and adequate reaction to correlations and clusters, a capability that has been previously reported in [2]. The paper presents a graph-based approach to partition database items across storage nodes so that most transactions only need to access one node. In this approach data items are represented by vertices, connected with an edge if they are accessed together by at least one transaction. Vertex weight is used to represent the absolute access frequency of an object, while edge weight represents the number of transactions that accessed both objects. The graph is then partitioned into k partitions by graph partitioning algorithm such as k-way min/cut [10]. This algorithm focus on minimizing the edge-cut (the objective) in a way such that the sum of the vertex-weight in each partition is the same (the constraint). The minimization of the edge-cut cost favors the deployment of vertices frequently accessed in conjunction in the same partition. On the other hand, the vertex-weight constraint is necessary to avoid the trivial solution of keeping all vertices in a single partition and paying a edge-cut cost of 0.

Hotspot Weighted Graph (HWG). The graph representation described is unable to represent geographical distribution, crucial for our system model. This is an important aspect as more populated areas may access more state items than others. In our scenario, partition imbalances are expectable and should be preserved if they represent the effective utilization pattern. To address this problem, the HWG algorithm adds a new "surrogate" vertex type which reflects the association of a state item to the surrogate. For the edge-cut algorithm, "surrogate" vertexs are indistinct from the remaining, with edge weight reflecting the number of transactions using the state item initiated in the surrogate. Surrogate vertices are given a weight large enough to force the partition algorithm to deploy one at each partition. All other vertices were assigned a weight of 0, to encourage the algorithm to focus on edge-cut cost and disregard the leveraging of the number of state items per partition (i.e. load-balancing).

To better illustrate this model, Fig. 1 depicts Twiter hashtags distribution by 4 surrogates (Lisbon, Porto, Braga and Faro) after HWG is executed (details are presented in Sec. 5). The figure evidences 4 large hotspots, each representing one surrogate vertex and the association of each vertex (the tweet hashtag) to the surrogates where the tweet was performed. Tweets at the center of the graph are connected to more than one location.

4.3 Deployment Revision Frequency

The frequency of execution of the deployment algorithm can have a non negligible impact on the system. Each run of the algorithm will necessarily require computing power at the oracle and have the negative impact on performance of migrating state items between surrogates. However, algorithm executions are crucial to ensure the proper deployment of the state items and reduce the number of transactions envolving two or more surrogates. In this paper, a moderate approach of running the algorithms after every 1000 transactions was adopted.

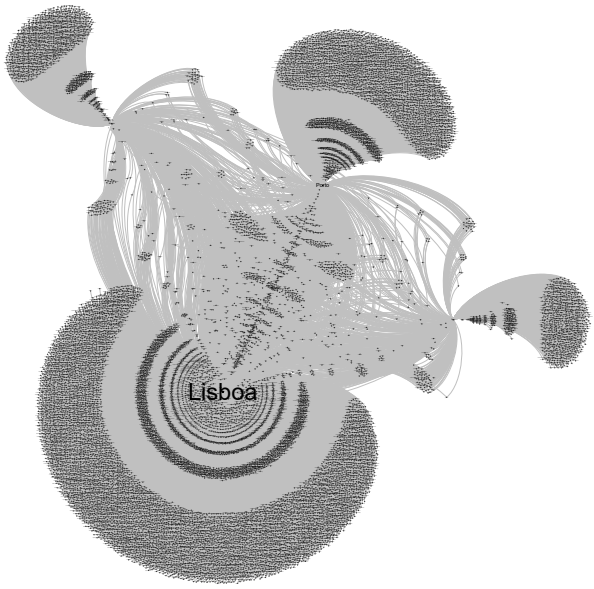


Fig. 1: Geographical object graph representation of a Twitter dataset from 2017 in Portugal.

It should be noted that the decision to use a frequency of 1000 transactions considers results observed in preliminary experiments with very frequent revisions. These experiments showed that as the revision frequency increases, so does the probability of the algorithms to follow inconsistent utilisation patterns, observed in the last few transactions. As a result, the number of state items migration increases exponentially, negatively impacting system performance.

5 Evaluation

To understand the impact of our state deployment strategies on system performance, we prepared an evaluation scenario using Twitter. The 53392 tweets used in this study were collected using Twitter's Streaming API,[3] between Jan, 10th and May, 1st 2017. The API was configured to retrieve tweets within 25km radius of 4 Portuguese cities (Lisbon, Porto, Braga and Faro). Supported by results in [11], it is assumed that the 1% of the tweets retrieved by the API are statistically representative of the hashtag diversity.

In total, 28615 distinct hashtags were found with 19120 of those only appearing once. Figure 1 is a graphical association between the hashtags retrieved and the cities where they have been published. The histogram on Fig. 2 shows that the majority of tweets (54%) only has 1 hashtag and that 80% of the tweets

[3] https://dev.twitter.com/streaming/overview

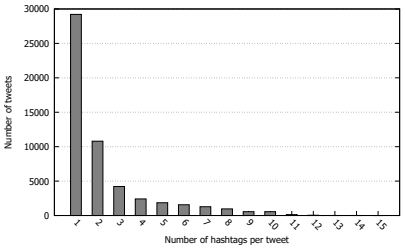


Fig. 2: Hashtags per tweet

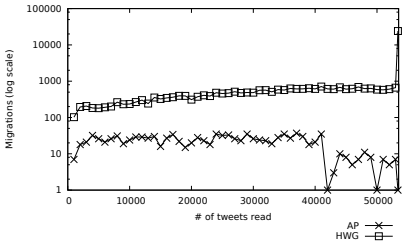


Fig. 3: State items migrations

have 3 or less hashtags. Overall, most of the tweets (66%) originated in Lisbon with Porto, Braga and Faro following with respectively 22%, 8% and 4%.

The evaluation consisted in running the algorithms described in Sec. 4 over an application state created from the tweets set. Each tweet was mapped on a transaction, with its hashtags representing the state items accessed. The transaction is assumed to have occurred on the location where the tweet was published and this location is associated to 1 of a set of 4 possible surrogates mapped on the cities. Each hashtag is initially affected to the location where it was first tweeted. Transactions are processed in chronological order.

5.1 Metrics

The performance of each policy was evaluated using 3 distinct metrics. The **number of state item migrations** and the **number of surrogates participating in a transaction** reflect the overhead imposed to the system. They are expected to be as small as possible in order to increase performance and reduce overhead. The **proportion of transactions enclosed in a single surrogate** complements the metrics above by highlighting the cases where the algorithm was able to achieve the optimal goal of deploying all the items in the same surrogate prior to a transaction.

5.2 Results

State Item Migrations Migrations are time and resource consuming operations, requiring the participation of two surrogates and possibly halting ongoing transactions. The goal of the algorithms is to be as efficient as possible by keeping this value low. The amount of group changes performed in the system is depicted in Fig. 3. The figure omits the Static and MtN algorithms. The former because no migrations are performed and the second because surrogate migrations are necessarily performed prior to each transaction and therefore results cannot be aggregated in the 1000 transaction slices.

The figure 3 shows a clear distinction in the performance of the AP and HWG algorithms. AP is considerably more conservative in what respects to items migration, approaching optimal results as time progresses. On average, AP achieves

an interesting result of 0.01 migrations per transaction. This result tends to indicate that once consolidated in a location, hashtags tend to be consistently more used in that location than in the remaining.

In contrast, HWG starts with an amount of group changes close to 0.1 per transaction and slowly increases during the experiment to around 1 group change per transaction. Results suggest that the creation of new edges or their update with new weights has a major impact on the graph partitioning algorithm, creating a map significantly different from the one defined in the previous revision. Part of this problem is attributed to the lack of memory of the algorithm, which focus exclusively on the actual state, disregarding the previous deployment. In practice this leads for example to the random breaking of ties, something that considering the overhead involved in state migration should be avoided.

Figure 3 highlights two abnormalities. The first happens at the 42000 tweets, after which the performance of the AP algorithm improves consistently. A careful analysis of the dataset revealed that 942 of the 1000 transactions in this period contain at least one of the hashtags *BBMAs* and *BTSBBMAs*. Such a large proportion clearly reduces the number of migrations to be performed as the number of tweets using the remaining hashtags was negligible. This trend continued until the end of the dataset, thus benefiting AP's memory approach. Interestingly, the behavior of HWG is orthogonal to this abnormally has it continued to shuffle hashtags unused in the period to leverage partitions. In contrast, the second abnormally, which occurred at the end of the dataset has a strong impact on HWG and can be explained by the small size of the evaluation window (392 transactions). It causes the graph to suffer from a temporary imbalance leading to migrations that could have been avoided with a larger and steadier observation window and confirms the inadequacy of high deployment revision frequencies.

Surrogates per Transaction Figure 4 depicts the average number of surrogates participating on each transaction. The plots of MtN and Static serve as control variables as they represent the theoretical minimum (recall from Sec. 4.1 that MtN moves all items to the same surrogate before every transaction) and the cost of not reacting to changes in utilization patterns. An higher average signals an undesirable situation negatively impacting performance on more transactions.

In the general case, HWG and AP show results between the two control algorithms, confirming that it is possible to improve the performance in scenarios like the one anticipated by our system model. HWG consistently exhibits an average below AP. This result is inline with our expectations, considering the additional effort performed by HWG during the revision deployment phase and which is clearly contributing for an improved geo-aware deployment.

Not surprisingly, the abnormality around the 42000 transactions discussed above can be equally observed in this metric, with all algorithms exhibiting a sudden spike in that region of the plot. The strong concentration of tweets using two particular hashtags is even sufficient for HWG and AP to surpass the average presented by the Static baseline test. To understand this behavior, it should be noticed that the hashtags must be deployed in one of the surrogates

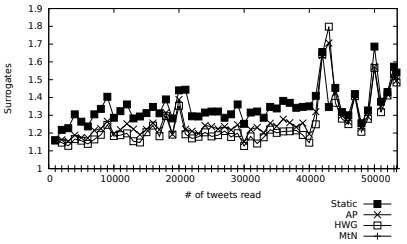


Fig. 4: Surrogates per transaction

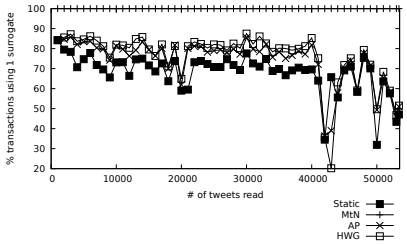


Fig. 5: Transactions enclosed in one surrogate

while a considerable proportion of the tweets are being requested by the remaining. These behaviors can only be mitigated by including in the experiments the datacenter meta-surrogate discussed in the Sec. 2 which would absorb these patterns by claiming the ownership of this hashtags. However, it should be noticed that both algorithms resume their performance once the exceptional scenario has been surpassed.

Single Surrogate Transactions The proportion of transaction that have all their items hosted by the same surrogate is depicted in Fig. 5. Expectations are that the geo-aware state deployment policies approach the perfect scenario presented by design of the MtN control algorithm.

Results confirm that geo-aware deployment policies are all able to outperform the results of the Static baseline test, with a similar gain of around 10% transactions. On average, HWG outperforms AP by approximately 1%. Not surprisingly, the proportion of transactions enclosed in a single group is also considerably hampered at the 42000-43000 transaction mark for the reasons discussed above.

Discussion A preliminary surprising result of the evaluation above was the good performance exhibited by the Static algorithm, capable of competing with other more informed approaches. However, this is a result whose applicability is limited as it strongly depends of the state utilization pattern. In particular, the results of Static show that in a fair proportion of the cases, tweeter hashtags are influenced by locality. The aggressive MtN algorithm equally presents interesting results. However, it cannot be directly compared to the remaining as their deployment revision frequencies are significantly different.

Overall, the performance of any geo-aware state deployment algorithm will be dictated by the trade-off between migrating state items between surrogates and performing distributed transactions, i.e., those involving more than one surrogate. Figures 6 and 7 materialize this trade-off in the results observed with the tweeter dataset from two distinct perspectives.

Figure 6 depicts a two-dimension perspective of the absolute number of item migrations against the absolute number of local transactions observed in our

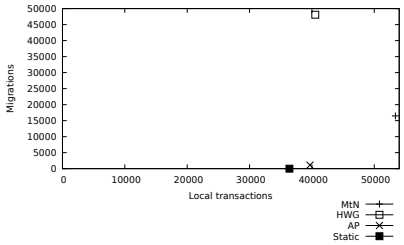


Fig. 6: Transactions in one surrogate per change

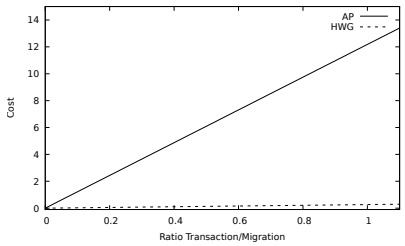


Fig. 7: Relation between algorithms

tweeter dataset. Best algorithms are expected to be concentrated at the bottom right corner of the plot, thus presenting the maximum number of local transactions using the minimal number of item migrations. Not surprisingly, champions on each dimension are the Static and MtN algorithms. However, as discussed before, they do not comply with the expected behavior.

The comparison between AP and HWG clearly suggests an advantage for AP. However, this is exclusively due to the excessively large number of item migrations required by the graph partitioning algorithm as HWG surpasses AP in the number of local transactions. A research direction that is clearly highlighted by this paper is the need to identify the constraints at the partition algorithm that refrain the number of migrations. In this respect, some form of "memory" that permits to preserve the association of items to a surrogate will be one of the topics to pursue.

In a real setting, the differences between these or other algorithms will necessarily be influenced by the effective cost of two operations: migrate a state item and the overhead of performing a distributed transaction instead of a local one. The materialization of these costs is outside the scope of this paper. However, Fig. 7 represents the overall "penalty" of AP and HWG for our experiment as a function of the ratio between the two costs. The plots represent the equation $\text{overhead}_a(x) = x(53392 - L_a) + \frac{1}{x}M_a$ where L_a and M_a are respectively the number of local transactions and migrations for algorithm a and x is the cost ratio between migrations and distributed transactions.

Again the figure highlights a clear disadvantage for HWG, with its overhead only being lower than AP when the ratio is very close to 0, i.e. when migrations have a negligible cost in comparison with distributed transactions. However, it should be noted that these results may be negatively influenced by the small proportion of transactions involving two or more state items in this dataset, as shown in Fig. 2.

6 Conclusion and Future Work

This paper have proposed and compared a new graph-based algorithm for the geo-aware state deployment problem. In contrast with the alternatives, the graph-

RSIGN: Secure Remote Qualified Signature Solution

Pedro do Vale[1,2], Carlos Cardoso[2], Renato Portela[2], and Ricardo Chaves[1]

[1] INESC-ID, IST, Universidade de Lisboa
[2] Multicert

Abstract. Qualified digital signatures (as defined in the EU Regulation 1999/93/EC[1]) are equivalent to hand written signatures. To assure this, they need to be created in a secure environment in order to grant them a strong legal value. The most common approach to accomplish this is with the use of secure devices, such as smart cards, to store and use the associated private keys. To avoid users having to carry such devices, this paper presents a solution providing remote qualified signatures on mobile devices. This is accomplished by having a remote service, able to create qualified signatures for remotely connected users. The implemented solution is able to provide a secure connection between the user device and the qualified zone in the server infrastructure, supporting multiple users each with unique private keys managed by an Hardware Security Module. This solution allows for qualified signatures in a scalable and secure manner, providing the creation of a commercially and technically attractive solution.

Keywords: Qualified digital signatures, Remote digital signatures, eIDAS.

1 Introduction

Nowadays many services are made available to the user digitally. Services such as e-Commerce and e-Government, allow users to conveniently carry out operations that otherwise would be made in person. These services, usually transactional, require privacy and integrity of sensitive data being transmitted and processed. To fulfil these security requirements, services rely on strong authentication mechanisms, such as the use of digital certificates and on digital signatures to assure authenticity.

Digital signature is a mechanism that relies on asymmetric encryption. It comprehends the knowledge of a private key, only known to its owner, and a public key for the users that wish to verify signatures computed by that owner. A signature is computed by calculating the hash value of the message to be sent, and encrypt that value with the private key. When receiving users get the signature along side the message, they obtain the hash using the public key, and compare it with a calculated hash of the received message. A positive match means that the message is authentic.

Usually public keys are stored in a digital certificate issued by trusted third-parties. This means that services and users need to trust these third-parties. Regulation 1999/93/EC[1] states legal requirements for qualified digital signatures and aims to achieve a high level of trust among users and services. This type of digital signature is created inside a Secure Signature Creation Device (SSCD) and is verified using a qualified certificate, that links signature verification data to signatory and confirms its identity. Due to the secure conditions to create qualified signatures, they are equivalent to handwritten signatures, granting them a strong legal value. One of the requirements for qualified signatures stated in the regulation is that the data necessary to create the signature is kept under sole control of the user, i.e. signature creation data cannot be located at a remote service. In a world where mobile devices usage is always increasing, this means that in order for users to create qualified signatures, users need to create them locally, thus implying the possession of the mentioned SSCD, such as a smartcards, which is not convenient since it is an extra hardware users have to carry.

Later, Regulation No. 910/2014[2], commonly known as eIDAS (IDentification and Authentication Services), replaced the previous one and adapted the requirements to current trends. This regulation does allow remote signature creation, as long as only the user can authorize signature creation in its name, i.e. it must not be possible for the service to create a qualified signature without prior authorization of the user. Since 2014, there has not been many solutions that allow the user to obtain signed data in a way that he does not need to carry a SSCD, which means that there is the opportunity for new ones to be proposed (such as [3][4][5], herein described). Usually, remote qualified signature solutions include an Hardware Security Module (HSM) as the SSCD. An HSM is a cryptographic hardware device that because it is an isolated environment and a dedicated device, allows for secure cryptographic operations (such as signature creation) and helps relieving the heavy load on servers. Usually these devices, have limited resources, such as storage, meaning that newly created data may be (and usually is) persistently stored externally.

This paper proposes Remote SIGNature (RSIGN), an HSM-based solution capable of producing qualified signatures whilst the user is remotely connected to the service, through its mobile device. For the user interacting with the service, the interaction is transparent, i.e. the HSM appears to be a local one. In this solution, the user must input its authorisation code, i.e. a Personal Identification Number (PIN), to authorize signature creation. One security advantage of this solution is that this authorisation code is never stored in the service and is only known to the user. This is accomplished by protecting users' signing keys using the HSMs master key and the users' authorisation code. To successfully sign data, one has to have access to the wrap, the HSMs master key (only known by the HSM itself) and the user's authorisation code (only known to the user).

The entities that compose the service and respond to users' requests is a frontend server that redirects requests/responses, a backend that deals with authentication, wrapped keys storage, certificates and communicates with the

Qualified backend, located in a restricted zone. That qualified backend communicates with the HSM for cryptographic data creation, such as qualified signatures.

The users interact with the remote service through a mobile application. When they register themselves in the service, a signing private key is generated in their name and they are then able to choose local document(s) to be (remotely) signed. Furthermore, the solution aims at being flexible, components are well separated and offer a REST API for inter-component communication. The HSM is accessible by an PKCS#11 interface which is a widespread standard intended for communication with cryptographic devices. This flexibility aims at being a commercial product, since it is being developed and is intended to be deployed at Multicert, an enterprise providing security solutions and certification services.

Document Roadmap

This paper is organised as follows: Section 2 states Related Work on the subject of qualified signatures, Section 3 presents the solution in more detail, Section 4 gives an insight on the Security Analysis and Recommendations for the proposed solution and, at the end, it concludes with some final remarks.

2 Related Work

This section starts by giving a more detailed insight on both European Union (EU) directives regarding qualified signatures[1][2]. Even though RSIGN proposes an interesting approach, other solutions capable of producing qualified signatures exist, they are also going to be discussed here.

Regarding mobile signing solutions, two main categories can be defined, namely: Server based and Mobile device based electronic signatures[6][5][7].

When discussing Server based solutions, the certificate which holds the public key necessary to verify the signature may be issued in the name of the user or in the name of the service provider to sign documents in the name of the user. In the latter, some authors argue that the user must give away its private key which goes against the requirements for qualified signatures because data is being signed in the name of the service provider and not in the name of the user, losing its qualified status[7] (and the non repudiation property by the user, for that matter). The same argument is stated in [5] and in [6]. This problem regarding the dubious status of qualified signatures when the signature is created at the server is justified with Art 2.2(c) of the EU Directive 1999/93/EC (the older Directive) on the requirements for qualified signature creation which states that *"it is created using means that the signatory can maintain under his sole control;"*[1]. However with the EU Directive 910/2014[2] release as a replacement to the previous Directive, this issue is clarified. eIDAS states that:

> "(51) It should be possible for the signatory to entrust qualified electronic signature creation devices to the care of a third party, provided that appropriate mechanisms and procedures are implemented to ensure that

> the signatory has sole control over the use of his electronic signature creation data, and the qualified electronic signature requirements are met by the use of the device.“

Plus, the aforementioned Art 2.2(c) has been updated to *“it is created using electronic signature creation data that the signatory can, with a high level of confidence, use under his sole control;“*[2]. This means that, with some guarantees on the way signatures are created, the user may delegate the signature creation to a service provider, as long as it can maintain control of when signatures are created, i.e. only with an explicit user authorisation, signatures may be created. Therefore, signatures created at the server may be eligible as qualified signatures.

Server based solutions usually recur to the use of a remote HSM to store users' keys. Tailored solutions[8], i.e meant for a specific environment, also exist. [4] presents a solution which is only suited to the electronic identification system of Austria, it is an example of a tailored solution. Zefferer *et. al*[4] argues that, in general, there is a lack of proper standardisation. This lack of common solutions makes it difficult for electronic identification of citizens to be recognized in different EU member states[9][10]. This is due to the national legislations that usually differ.

On the other hand, mobile device electronic signatures imply some signing capabilities at the client side. Many of this solutions are based on smart cards. Even though smart cards are robust devices for storing users' private keys, usability wise, they are not the ideal approach, since it requires the user to have a smart card reader device[8][7].

Martínez-Montesinos *et. al*[6] argue that the main disadvantage of Mobile device based signatures that require the user possesses a mobile phone, is that the mobile network carrier (also known as mobile operator) has to establish an agreement with the entities that provide the signing capabilities which is not the case for most network carriers.

In [11], the authors argue that usability should be an important requirement when designing qualified signature solutions and that qualified signature creation at server side is ”user friendlier”, since the the user only authorises the signature creation and the rest of the process is performed remotely by the server.

Regarding similar solutions, in [3] a remote HSM-based solution is presented. It protects the signing key by wrapping it with the HSMs master key. It also protects the wrap with the user public key. The private key corresponding to this public key (different that the signing key pair) is protected with a key derived from the user PIN. To fulfil a sign request, after a correct PIN insertion and Short Message Service (SMS) authorisation code sent by the user, the private key is decrypted by a PIN-derived key and the wrap is obtained by decrypting it. The wrap is delivered to the HSM that unwraps the signing key and then the signature is created. Authentication of the user is done by the network operator. The user must possess a Subscriber Identity Module (SIM) card for successful authentication. This solution depends in the external network operator and the possession of the SIM card. Also, some of the trust must rely on an entity outside the HSM since the security the PIN is verified out of its scope.

Zefferer *et. al*[4] propose a remote HSM-based solution that makes use of Austrian provided modules for qualified signature creation. It relies on the use of a widely used technology for user interaction, the SMS. However the user must input a Transaction authentication number (TAN) that is sent through in the SMS in plaintext. Despite being a convenient solution, it may not be secure. Plus, like in [3], it relies on the network operator as the it is required by the SMS technology.

Rossnagel *et. al*[5] propose a client-side signature creation that relies on a SIM card with signing capabilities, which is not common in SIM technology. To mitigate the identified problem of business lock-in (there would have to be changes in the mobile network infrastructure), certification is left to a third-parties, chosen by the user. Physically, the user would have to buy this specific card, activate signature data and require a certificate to be issued. Usability wise, this solution lacks the convenience remote signature offers.

To conclude, most solutions rely on external technologies for either authentication or signature creation, such as SIM cards. This means that there is always an hardware dependency, which should be avoided to achieve platform independence. Furthermore, in the case of remote signature solutions, authorisation of signature creation in the name of the user is always a risky procedure because it requires the authorisation data to be sent over the network. Communication channels must guarantee the authenticity, privacy and freshness of this data. Since EU Directive 910/2014[2] there has not been many solutions that could potentially be used by the general community, giving opportunity to explore innovative solutions.

3 Proposed Solution

Given the requirements set towards a service capable of remotely signing documents and the existing solutions, herein a dedicated service is presented. This solution avoids the possession of extra hardware on the client-side. It includes an Hardware Security Module (HSM) to perform critical security operations.

3.1 RISGN's Architecture

The architecture depicted in Fig. 1 presents the overall components of RSIGN. On the client side, the user accesses the service through its mobile device, such as a smartphone or a laptop, properly connected to the internet. Operations triggered by users' are redirected to the Client Portal, the frontend of the system. This component simply redirects requests/responses between the user and MTC Backend. There, the client may login. This authentication is a necessary step for the RSIGN core operations. After successful authentication, operations are redirected to the Qualified Backend, a dedicated secure server. From there, requests to the HSM are made, and responses are forwarded. As mentioned before, the limited storage capability of the HSM drives the need to store signing keys externally. However, these keys can never be visible in plaintext outside the

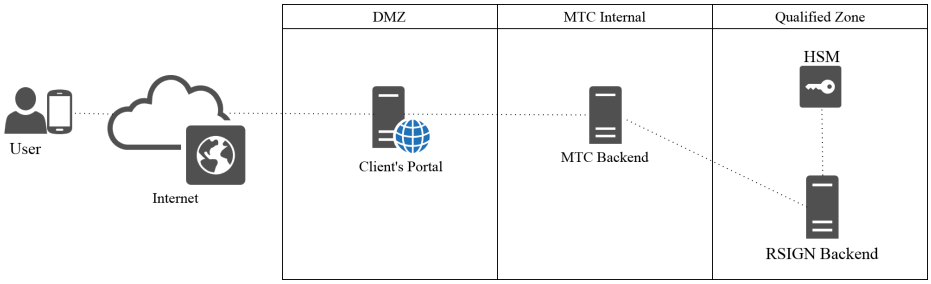


Fig. 1. RSIGN's Architecture

boundaries of the HSM or else it might be compromised. This is accomplished by protecting the user's private signing key, with a wrapping key that can only be computed using the HSM's master key and the user's PIN as input.

To reflect the requirements of user authentication and key storage, the MTC Backend is in fact composed of a key database (where key wraps are going to be stored) and a authentication database (to store users' credentials), as illustrated in Fig. 2.

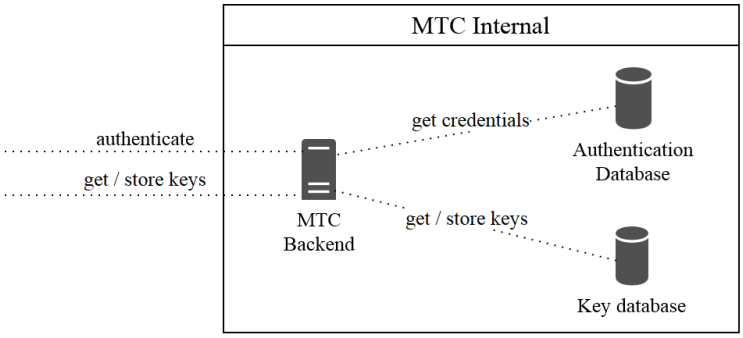


Fig. 2. Components in the MTC Internal zone.

Since this is a security solution, it is important to group the architecture's components into zones with well defined boundaries between them because the data that enters and leaves each zone needs to be protected. For instance, the PIN must must not be disclosed, i.e. it must not be accessible in plaintext (in the user endpoint and Qualified Zone exempt). This facilitates compliance with eIDAS[2] in the sense that only the user has the necessary knowledge to trigger qualified signature creation and no potentially less secure components (e.g. Client Portal) can get access to sensitive data in plaintext. Furthermore, this zone separation allows for a modular deployment of the components in the physical infrastructure.

3.2 RSIGN's secure channels

In order to achieve a secure solution, the communication going inwards and outwards from each component must be protected in some way. The Transport Layer Security (TLS) standard provides the security properties that protect the messages exchanged between components that are directly connected. However, regarding the eIDAS[2], it must be assured that sensitive data (in this case PINs, data to be signed and signatures) exchanged between the user and the components in the Qualified Zone is protected.

Fig. 3 illustrates the secure communication channels between the several system components. A TLS connection among communicating components and a secure channel on top of the TLS channels is established before any sensitive data is exchanged.

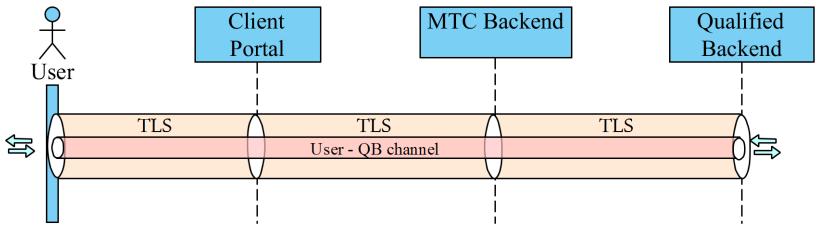


Fig. 3. Communication channels between system components, before and after login

In RSIGN, the establishment of a secure channel that links the user endpoint and the Qualified Backend is initiated by the Diffie-Hellman (DH) Key Exchange protocol, at user log in. After the user successfully authenticates itself towards the system, the user's DH public values are sent to the Qualified Backend and once the Backend computes the shared secret, it sends its own public value so that the user may obtain an equal shared secret. Afterwards, the user and the Qualified Backend exchange random data to confirm the use of the newly created link/channel. From that moment on, the user and the Qualified Backend share a secret key that may be used to protect data that not even intermediary components, i.e. the Client Portal and the MTC Backend, can have access to in plaintext.

The Diffie-Hellman (DH) protocol[12] outputs a shared secret key that provides Confidentiality to the data being exchanged. However, this protocol lacks Data Authenticity and Freshness mechanisms. Therefore the Hashed Message Authentication Code (HMAC)[13] mechanism and the use of *nonces*, random sequence numbers, are added to assure the aforementioned security properties.

After the completion of the DH protocol, messages are exchanged in the following way: the sender that wishes to send a message, M, derives an encryption and integrity keys, K_c and K_a, from the agreed shared secret key, K_s, and

computes the HMAC of the encrypted data plus *nonce*, N_1, to be sent. It then sends the encrypted message and the HMAC alongside it (see (1)).

$$\begin{aligned} K_c, K_a &= derive(K_s) \\ A &= \{data, N_1\}_{K_c} \\ M &= A, HMAC(A, K_a) \end{aligned} \tag{1}$$

The receiver then derives the encryption and integrity key (the same way the sender did) and computes the HMAC of the encrypted message to verify the authenticity of the data. After decrypting and processing the data, it responds to the sender with a new message, M', containing response data, $data'$, and the incremented *nonce*, $N_1 + 1$ (see (2)). If the original sender verifies that the message is authentic and the nonce is as expected, it accepts the response.

$$\begin{aligned} B &= \{data', N_1 + 1\}_{K_c} \\ M' &= B, HMAC(B, K_a) \end{aligned} \tag{2}$$

3.3 RSIGN's operations

The operation RSIGN provides are: *Data Generation*, *Signing* and *Change PIN*, which are detailed below. Before each operation, the user must log in the system, using its email and password. This means the first interaction the user performs in RSIGN is signing up, using its email and password.

Data generation The following protocol, depicted in Fig. 4, represents the *data generation*, i.e. signing key and PIN, necessary for the *signing* operation. In this diagram, the user requests its intention to be able to sign data in the future. For that, a key pair and a PIN is generated by the HSM. Although, as stated before, the private key must not be return in plaintext outside the HSM. For this, the newly generated private key, $K_{signing}$, is wrapped using a derived key (see (3)). This wrapping key, $K_{wrapping}$, is derived from the master key of the HSM, K_{MK} (which never "leaves" its premises), and in the derivation process, a hash of the PIN is added as a computation parameter. After the operation is complete, the wrapped key, the public key and the PIN are returned. The key wrap is signed by the Qualified Backend for future authenticity verification. Private and public key pair, PIN and wrapping key are deleted from the HSM, where all this operations took place. Note that the PIN is returned to the Qualified Backend in plaintext. This is due to feature limitations of the HSM and is only secure because the Qualified Backend is trusted, to that extend.

$$\begin{aligned} K_{wrapping} &= derive(K_{MK}, hash(PIN)) \\ key\ wrap &= wrap(K_{signing}, K_{wrapping}) \end{aligned} \tag{3}$$

In the Qualified Backend, the PIN and the wrap are encrypted and, along the public key, sent to the MTC Backend. The PIN is then delivered to the end user,

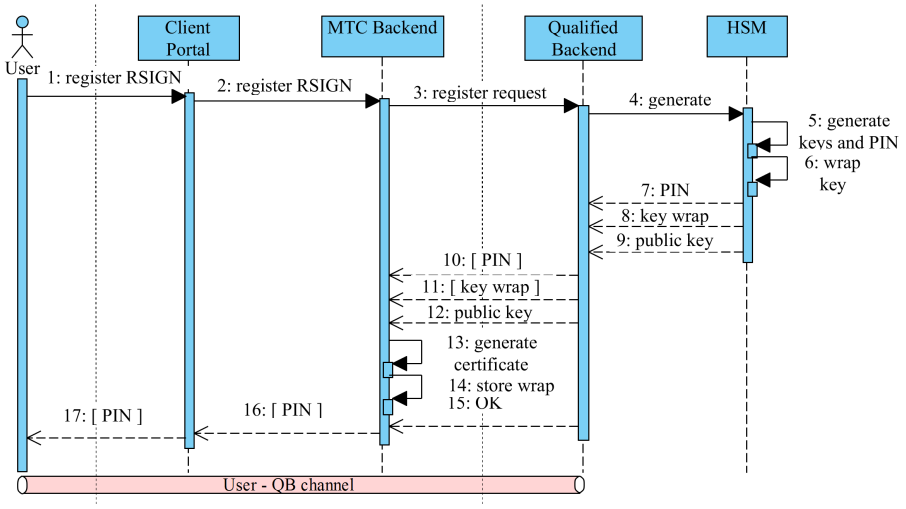


Fig. 4. Sequence digram presenting user data generation in RSIGN

using the secure channel, where the PIN is obtained. The user then becomes the only component to have knowledge of the PIN. It is not persistently stored elsewhere.

This operation can be compared with the proposed solution in [3] in which the master key is used to wrap the signing key. In RSIGN the signing key is wrapped with a temporary key, which is destroyed after the operation and it needs to be computed again to obtain the singing key to be able to used it thus making the HSMs master key less exposed since it is not used directly to wrap keys.

Signing Following this *data generation* interaction, RSIGN is ready to sign data in the name of the user. Diagram in Fig. 5 states a signing operation. Again, after a successful login, the user sends in the hashed data in the sign request through the secure channel, along with the authorisation PIN. After receiving the request, the MTC Backend retrieves the wrap of that user's signing key and sends a request to the Qualified Backend. Then the HSM performs the key unwrapping (the same way it was computed in 3). Unwrapping operation fails if the given PIN is incorrect (MTC Backend blocks the user after 3 unsuccessful PIN tries for a limited time). From there, the private key in the HSM is used to sign the data in the signing request. Again, private and wrapping keys are deleted from the HSM and the qualified signature is returned to the end user via the established channel.

Change PIN The user may also change its PIN and it is advised that he/she does so after it receives the PIN in the *Data Generation*. Similarly to that initial operation, for this to happen the old PIN is used to compute the wrapping key,

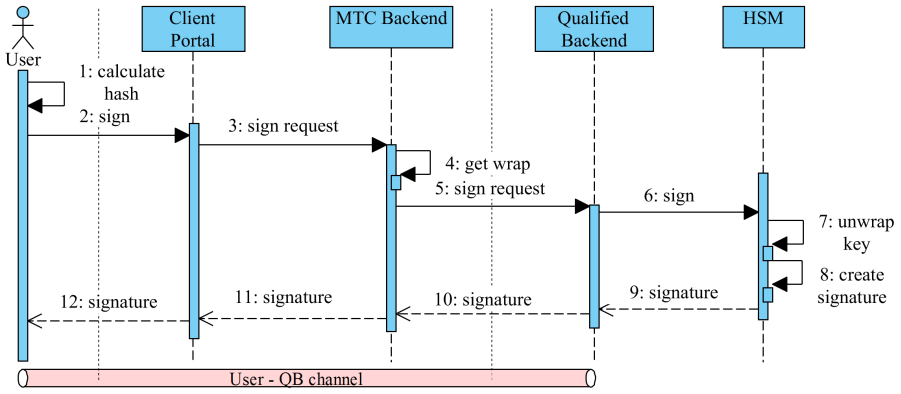


Fig. 5. Sequence diagram presenting a sign request

in the HSM, which then is used unwrap the singing key, compute a new wrapping key with the new PIN and wrap the singing key to be stored externally. Again, the unwrapping fails if the old PIN is incorrect.

3.4 Implementation

As mentioned before, the solution is being developed in collaboration with Multicert. This solution is meant to be deployed in an existing signature solutions which offers various options for signing. That solution includes an adapted version of the one herein presented. For simplicity sake, some components were not implemented in this prototype since they are already deployed in Multicert's infrastructure. The core of this solution consists of the Qualified Zone and its interaction with users. Therefore, the implementation consists of a user with its smartphone connected to the Internet, interacting with RSIGN through an Android application and using a Representational State Transfer (REST) interface, which allows the user to signup, login, request signature *data generation* (which generates the signing key and PIN, request remote signature creation of one or more files and request its PIN to be changed. At the server side, the Client Portal relies on an Authentication Database for user validation (at the MTC Backend) and communicates directly with the Qualified Backend redirecting requests/responses. There, the Backend, connected to a key database (to store key wraps), communicates with the HSM for the required operations.

The communication channel between the application and the Client Portal is protected with Secure Socket Layer (SSL). The communication channel between the Client Portal the Qualified Backend, accomplished through SSL also, requires mutual authentication. This means that each of this component must have a valid certificate and a way to store them. In both cases, certificates and keypairs are stored in a trust store and a key store, respectively.

Regarding the Qualified Backend - HSM communication an PKCS#11 interface is used. PKCS#11 is a widespread standard for communication with cryptographic devices which promotes the flexibility of changing the HSM to be installed. It requires the Qualified Backend to be authenticated towards the HSM.

4 Security Analysis and Recommendations

The security analysis to the Proposed Solution is presented in this section, considering what needs to be considered when using or extending it.

Since this is a remote service, the security properties of communication the channels that link the various components of the system is an important issue. An attack on the data being transmitted may pose a serious threat to the service. Mutual authentication and confidentiality between the Backends and between the MTC backend and Client Portal should be ensured by the SSL protocol using up-to-date qualified certificates.

As for the secure communication between the user endpoint and Qualified Backend, it is accomplished by an indirect channel which applies the Diffie-Hellman (DH) Key Exchange protocol, the HMAC mechanism and the use of *nonces* to assure Confidentiality, Authenticity and Freshness to sensitive data, such as PINs, signatures and hashed data. This allows for data to be sent in and out of this two zones without having to trust the information in plaintext to intermediate components. Note that, to avoid impersonation attacks, the public value the Qualified Backend sends to the user endpoint in the DH protocol, should be signed. For this, the user must have access to the Qualified Backend public key to verify the signature.

In terms of hardware, being the core of this solution, the qualified zone must only include certified components. Specifically, the HSM and the Qualified Backend must be secure and trusted. Without this guarantees, the solution cannot be considered a qualified service.

Regarding the HSM not storing the PIN anywhere and needing it to compute the (un)wrapping key in the HSM, it is a great advantage in granting the qualified status of the solution. On the other hand, it has one clear drawback: when the PIN is incorrect, it means that the validation must go from end to end of the components for trying again. This is a trade-off between Performance and Security, wherein Security is considered a priority. Also, the number of times the user may attempt a PIN should be limited, to avoid brute force-attacks.

To protect data meant to be signed, it is recommended that the hash of that data is sent. This means that the hash must be computed at client-side, which requires a mobile device with such capabilities. It also means that it is not in the HSM secure environment that hash values are computed, using its ”bullet-proof” inward algorithms. This brings to question the environment in which the client is. In the implemented solution, for instance, it is assumed that the android application, and the smartphone where it is hosted, is secure. Nonetheless, one advantage of this solution is that the service offers a flexible interface, i.e. porting to a different platform is possible and not difficult.

The required initial steps, signup and login, are not the most relevant feature in the solution. This is due to the fact that there is already strong authentication mechanisms, this paper does not propose an innovative way of user authentication.

Lastly, given the limited strength of the PIN, an alternative should be implemented. Future work includes a way to compute a user friendly authorisation code that is transformed and send to the Qualified Backend. At the moment, this is mitigated by the use of the mentioned secure channel.

5 Conclusion

Being able to create signed data remotely is useful for mobile users but it requires the signature creation to be created in a secure manner considering that this creation might be hindered by the network vulnerabilities. Qualified Signatures were first introduced in Directive 1999/93/EC[1]. They must be created by a Secure Signature Creation Device (SSCD), and the operation must only be authorised by the end user.

Most of the related solutions that in some way are capable of performing Qualified Digital Signatures consider user dependency of extra hardware. Solutions independent of extra hardware promote usability, there have not been many solutions of this kind proposed.

The presented solution, Remote SIGNature (RSIGN), considers different security zones in which components are distributed. It includes an Hardware Security Module (HSM), as the SSCD and a database for persistent storage of signing keys. The need for a database is driven by the limited storage capability of the HSM. But, to achieve the requirements in the current regulation[2], the private key may not be in plaintext outside the HSMs domain. For that reason, the private signing key is protected by both user and HSMs private data. The user private value, a PIN, is never stored in the service, meaning that only the user may authorise qualified signature creation.

The proposed solution is a secure and remote environment where qualified signatures may be created in the name of the user. The strength of this solution is in the secure and scalable way the users' signing keys are managed as well as how protected the data that users' send and receive during transmission.

Acknowledgments

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

NFC-based, off-line door lock access control mechanism

Miguel Azevedo[1], Rui Espinha Ribeiro[1], André Zúquete[2]

[1] Universidade de Aveiro,
{lobaoazevedo,ruiespinharibeiro}@ua.pt,
[2] DETI / IEETA, Universidade de Aveiro,
andre.zuquete@ua.pt

Abstract. This paper proposes an NFC-based, off-line door access control mechanism. The credentials to access the door services, such as opening its lock, are first obtained by users from a central authentication server. Those credentials are time bounded, i.e., they can only be used on a specific time frame. In our case, we used daily credentials, but other time intervals could have been used. Door locks keep a record of all successful opening requests, which can be collected using another door service, also available through its NFC interface. The access credentials are fetched and used, within a challenge-response authentication protocol, by an application running on an NFC-enabled device, such as a smartphone. For a proof of concept we implemented a prototype of our system with an Arduino with an NFC shield, for controlling a door lock, an Android Nexus 7 mobile device and a Web server for providing daily access credentials.

Keywords: NFC, offline access control, role-based access control, mobile communication

1 Introduction

NFC (Near Field Communication) usage in mobile and embedded devices has been growing over the last decade in various contexts, allowing portable short-range communication with a simple setup between two devices. One interesting use case for such technology is to develop access control systems, in which the authentication process is performed in a quick and effective way. Such systems typically consist of an NFC-enabled, embedded system which relies on network access for user authentication and access granting using remote servers. However, this type of systems are exposed to a number of security vulnerabilities which might not be directly tied to the NFC system, but rather to their network interface. Furthermore, the requirement of a network connection is a undesirable, as it increases installation costs.

In this article, we present an NFC-based access control system for a door which does not depend on a network infrastructure to preform user authentication and access granting. Furthermore, the credentials presented by a user

are not static, such as the fixed, low-level identification numbers presented by NFC tokens (Unique permanent Identifiers, UIDs). Instead, credentials are time-bounded secrets that are explored in challenge-response protocols conducted with doors. Users rely on NFC-enabled personal devices, such as smartphones, to keep their credentials and to conduct such protocols with the doors of interest. Those credentials are user-specific, daily authentication keys, deterministically derived from (i) a door master key, previously assigned to the door, (ii) the authorized dates and (iii) the user identity. The key derivation process is based on the HMAC-based key derivation function (HKDF [5, 4]). The identity of the user accessing the door lock allows the door micro-controller to log the actual user that performed the access attempt. Doors have different master keys for different services, such as lock opening and maintenance.

For a proof-of-concept, we successfully developed a prototype in which a door lock, controlled by an NFC-enabled micro-controller (Arduino) with no network access, grants access to a room or area upon a successful NFC interaction with an NFC-enabled personal device (Android Nexus 7 tablet). The provisioning of daily keys was implemented by a Web service.

This paper is structured as follows. Section 2 presents the related work. Section 3 presents the system architecture. Section 4 presents our prototype implementation. Section 5 presents a security analysis of the proposed system. Finally, Section 6 presents our conclusions and highlights some future work.

2 Related work

In a brief overview, all works in the area of NFC communication and access control systems that can bear significant resemblance with this project mostly share some common features: they are based on a Microcontroller Unit (MCU) integrated in the door lock to provide authentication when coupled via NFC with a smartphone or smartcard; this MCU is usually connected to a remote Web server which handles most of the authentication process. Also, no solution with an authentication scheme similar to ours was found, as well as no other solution with an off-line access control door lock system. The remaining of this section provides specific comparisons with the work we are presenting.

In [9] the authors propose a system composed by an RFID reader, incorporated in the door lock, connected to a computer. This computer verifies the UID provided by an RFID tag when it is near the reader. This work is quite different from ours, as a network connection is always assumed. Also, its authorization scheme relies merely in verifying an UID, which can be stolen and cloned in some other device.

In [2] the authors focused on developing a smart door lock with a PoE (Power over Ethernet) driven circuit and UDP remote communications. Such as the previous one, it provides authentication based only on a simple UID verification.

In [11] the authors presented a system also requiring network access to handle user authentication. However, this implementation explored information hiding techniques to transmit information over the network: an access code was em-

bedded into an image which was selected by the user so it could be sent to the server for code extraction. The server then compared both access codes to provide authentication. The disadvantage of this solution when compared to ours is also the door lock system's network dependency to provide authentication.

In [8, 7] the authors present an authentication process using NFC-enabled smartphones that is more complex than the aforementioned, although it does not add any extra step to deal with user interaction. The smartphone carries an RFID tag UID, which is provided to the door lock encrypted with the system's public key. The door lock system forwards the encrypted data to a remote server which handles the decryption process and validates the UID. This authentication process clearly differs from our solution, as the MCU relies on a network connection.

In [1] the authors present a solution that, Unlike the previous ones, features a user authentication step: a password typing in the user's smartphone. Their solution uses an hardware module that is also included in our proposal, an RTC (Real Time Clock), for getting timestamps in each device coupling event. Such timestamps are provided to the MCU and combined with the introduced password to obtain a numerical sequence which is compared with the received sequence. Similarly to our proposal, this system does not a imply a network connection but it imposes a password typing, which is not only slightly inconvenient (one of the main advantages of NFC authentication is its nearly instantaneous communication process) but it may be vulnerable to a keystroke logging system.

In [3] the authors proposed a central control unit, within the home network in which this system was deployed, which was connected via UTP cables to the door locks. After a user establishing a Wi-Fi connection with a central unit with its smartphone, its previously assigned key is encrypted and sent to a remote server, which performs the verification process. This system is highly dependent on the Wi-Fi security and does not rely on NFC in none of its authentication steps, although it can represent a relevant proposal in the field of smart door locks.

3 Architecture

The system's architecture is presented in Figure 1. It is composed by (i) a central authentication server, (ii) users' mobile devices equipped with a network interface (Wi-Fi or 3G/4G), NFC communication and running a special application and (iii) NFC-enabled doors locks, managed by a micro-controller.

3.1 Authentication Server

The authentication server keeps a set of user profiles and a set of door master keys (two per door, master maintenance key and master door opening key). The user profiles contain identification and authentication information about users, as well as doors' access clearance policies (can/cannot open, can/cannot

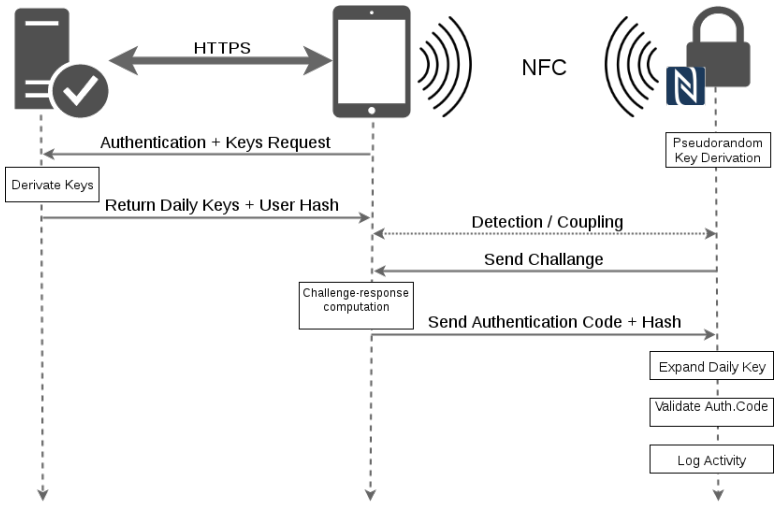


Fig. 1. Architecture and high-level interaction protocols of our NFC-enabled door opening system

manage, etc.). Profiles also contain the set of days for which access credentials were provided to the user, for specific doors.

All master keys of doors are shared with the correspondent doors. Their value is random, thus different doors will have different keys. Although this may complicate the setup of the system, it protects the whole system from the compromise of a door lock, since its keys do not reveal anything about secrets from the rest of the system.

The communication between users and servers is based on an HTTP API, protected by an SSL transport. The server's API is meant to be explored by specific applications running on users' devices, not by ordinary Internet browsers.

The main functionality of the central authentication server is to provide applications running on users' mobile devices with the necessary daily keys for requiring services to specific doors. Such daily keys are generated from master keys with the following key derivation algorithm (see Figure 2):

$$\text{daily key} = \text{HKDF}\left(\text{HKDF}\left(\text{master key}, \text{current date}\right), \text{user hash}\right)$$

where HKDF stands for an hash-based key derivation function [5, 4]. The user hash value is a fixed-size hash of a variable-size user identification record managed by the authentication server. This value is provided to the application in the mobile device when it gets the daily keys.

3.2 Users' mobile devices

The users' mobile devices must be equipped with a network connection interface (Wi-Fi or 3G/4G) and NFC communication. The network communication is

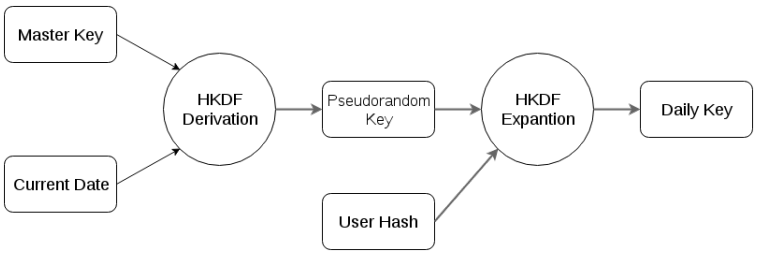


Fig. 2. Derivation of a daily key from a master key, a date and a user identification hash

fundamental to interact with the authentication server, in order to get credentials to access specific doors. The NFC communication is fundamental to interact with those doors.

The mobile devices run a specific application to deal with the opening of doors. Users use this application, and the network communication, to interact with the authentication server in order to fetch credentials (daily keys) for accessing specific doors in a set of given days. Then, upon presenting their mobile device to the NFC-enabled door, the application gets activated and simply asks if an authentication protocol is to be conducted with the nearby door. Note that all the credentials to the authentication protocol are selected automatically, since there is no ambiguity: the day is the current day in the user's mobile device, the door identity is provided by the door itself (through NFC) and the user identity is stored along with the daily keys.

3.3 Door lock

The door lock is an NFC-enabled device that does not need to have a network connection: it uses only an NFC interface.

We will assume, hereafter, that in the setup of each door there was a setup procedure that enabled both the door and the authentication server to share a door identification and a set of secret keys. The details of such procedure are not relevant for this paper.

The door supports two different services. One is the door lock opening service, the one that first motivated our work;the other is a maintenance service, which can be used to manage the door thorough the NFC interface. We designed a simple and specific maintenance protocol, which can be explored for some housekeeping actions that will be detailed later on.

Services are selected by the application running in the users' mobile device. The selection happens upon the authentication process (whose phases are represented in Figure 1) and dictates a specific selection of a daily key. The user needs to select the door lock (represented by an unique identifier) and the operation (s)he aims to perform (which for non-maintenance operations corresponds to an access request; maintenance-related operations are only suggested for selection

if the user has permissions to perform them). For the door lock opening service, it needs to use a daily key derived from the master lock opening key; for the maintenance service, the application needs to use a daily key derived from the master maintenance key.

3.4 Authentication Protocol

A mobile device is required to authenticate itself to a door before requiring a particular service from it. The authentication protocol is multilateral, i.e., both the device and the door lock authenticate itself.

The door lock is required to have some human-readable, visible identification string, which is also provided through NFC to the mobile device. The door lock identifier (some unique string of characters) is used to select the appropriate daily key from the user's set of daily keys, and is also presented to the user, in the mobile device's interface, for confirmation before continuing with the rest of the protocol. This method is enough to authenticate the door and to prevent the authentication protocol to run without the user consent, e.g. triggered by some nearby attacker.

The authentication of the application running in the mobile device is performed with a response computed from a (random) challenge provided by the door and a door daily key:

lock $\longrightarrow$ app : door id, challenge
app $\longrightarrow$ lock : uinfo, HMAC(dkey, challenge), service

where `uinfo` is the user info required the compute the correspondent user hash and `dkey` is the user's daily key for the `door id` door. This mechanism is represented in Figure 1 in the "Send Authentication Code + Hash" phase.

The door uses the appropriate master key, given the service requested, its current time value (with a daily granularity) and the user hash (which is calculated using the provided user info) to compute a daily key and validates the received response for the challenge provided. If the response is correct, the user identification information and the current time (with a smaller granularity, say, minutes or seconds) are logged in the door's controller memory and the service is granted. In the case of the lock opening service, the door opens; in the case of the maintenance service, its provisioning involves a subsequent communication between the door and the mobile device.

3.5 Maintenance service

The maintenance service exists for configuring operational parameters of a door controlling system. Example of such parameters are its identification, its keys, the current date, etc. Furthermore, the maintenance service can be used to retrieve and clean the list of users that actually opened it in the past.

This maintenance protocol should be made available to all users that authenticate themselves using daily keys generated from the door's maintenance master

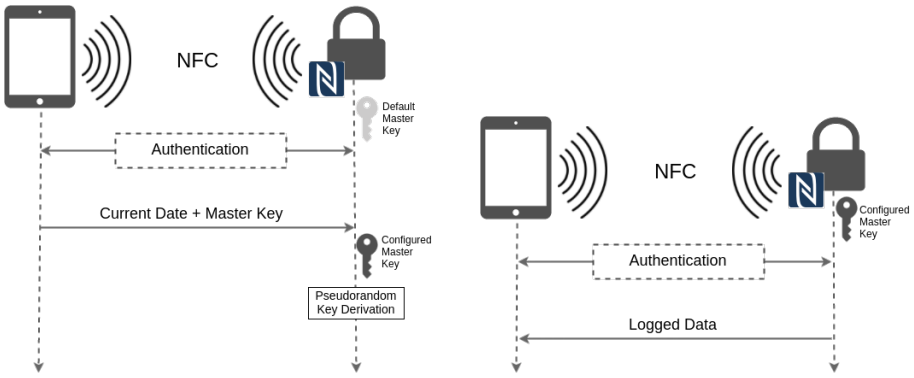


Fig. 3. Configuration and maintenance protocols performed by an administrator: configuration of the door lock date and master key (left) and retrieval of a door log system, containing all successful authentication operations (right).

key. This enables different users, i.e. users with different identities, to perform maintenance operations on doors, without requiring doors to know in advance the identities of such users. The access of users to maintenance operations is defined by the authentication server, when providing daily keys computed from a master maintenance key or a master lock opening key.

Administrators can set master keys on locks and retrieve their logs for the last authentication events (see Figure 3). Each log entry contains every successful access request event as well as the correspondent response and the user ID. Similarly to the configuration process, administrators authenticate for log retrieval using keys derived from the configured master key. Besides user and time information, the type of operation and the current state of the door lock are also included in each log entry - the operation can be a simple door lock opening request or a maintenance operation and the door lock's state indicates whether the door lock lacks configuration, if it is administratively locked down, if it is in log retrieval mode or if it is simply "running" (awaiting for any access request).

4 Prototype implementation

The three main components which support this system were implemented mostly using widely available and well-known hardware and software. The implementations of such components are described in this section.

4.1 Authentication Server

This server provides a Web service API for all the interactions between itself and users or administrators (which use the application in a mobile device). It was

implemented in Go[3], which provides a standard library supporting all the Web service implementation and cryptographic operations. The server uses a relational database for system metadata storage, which was responsible for the single third-party package that used on this implementation: the DBMS (Database Management System) driver. The PostgreSQL database management system was chosen for this implementation, mainly for its simplicity and scalability.

4.2 Android Application

The Android applications supports two user roles: client and administrator. The first one can view all daily keys already fetched keys and their associated doors, and authentication to a door system via NFC. On the other hand, the administrator can configure the door lock and retrieve access logs from a door lock upon authentication.

As previously mentioned, the application uses the server's Web API for all remote operations. Relevant data is stored for later off-line access. For this purpose, the application uses an database which stores user profiles and allows secure access to all relevant data. The chosen DBMS was SQLite, as it is lightweight and suited for portability. The entire application was developed using Android's native APIs.

All HTTP requests to the authentication server are performed over HTTPS connections and the user's authentication is performed using a user plus password basic access authentication. When a daily key is provided, it is stored in the local database, along with the user hash and the correspondent date.

Upon NFC-coupling, the authentication protocol is performed with NDEF messages.

4.3 Door Lock

As already mentioned, the door lock consists of Arduino board, coupled with a SeedStudio NFC shield and connected to an electrically activated door lock strike. Since the door system needs to preforming fairly computationally complex operations for authentication, such as key derivation and expansion, the Arduino M0[4] model was chosen, since it is based on an ARM Cortex-M0 microprocessor with an on-chip RTC (Real Time Clock).

For the software implementation of the challenge-response authentication protocol (namely hashing and message authentication code functions) the ArduinoLibs[5] API collection was used with the addition the HKDF implementation. All the NFC communication was handled by the SeedStudio API, improved by Elechouse[6]. The remaining features of this component, such as interruption driven behavior and real time clock setup, rely on the support provided by the Arduino native APIs.

[3] https://golang.org/
[4] http://www.arduino.org/products/boards/arduino-m0
[5] https://github.com/rweather/arduinolibs
[6] https://github.com/elechouse/PN532

4.4 NFC Interaction

All the described interactions between the doors and users' mobile devices take place over an LLCP (Logical Link Control Protocol) session, which is typically initiated by the door system. This happens due to a software limitation imposed by the used SeedStudio NFC library. Every data element exchanged between those two devices is transmitted using NDEF messages. The Android environment only provides means to identify the initiator in case it uses a Google Play Store identifier for the initiator application (this is on a scenario where both devices are Android based), therefore this Android application must be active during coupling.

Upon coupling, the door controller acts as initiator, opening the LLCP session and sending the door identification and a randomly generated challenge. The Android application, as a target, reacts to the reception of this information from the door and, upon a consent given by the user to the door identity, computes the corresponding response and returns it, as already described. An authentication and maintenance protocol was outlined, in order to support multiple operations on the door lock based on a user's role. This protocol is extensible to multiple users. In this prototype, two roles were considered.

General users

Users who assume this role are mainly allowed to perform interactions with a door lock without an administrative role. In this case, the only possible operation is to request access through the specified authentication method in this article. As aforementioned, a challenge-response is mechanism is triggered via NFC: the door lock system sends a randomly-generated challenge (a *nounce*) in an NDEF message to the Android application, which is replied with another 69 bytes-long NDEF message - 4 bytes specifying user information in order to enable daily key-derivation by the door lock system, 64 bytes containing the authentication code (which results of the challenge-response computation by the app as described in 3.4) and a byte indicating the desired operation (which in this case is to request door opening to the lock).

Administrator

As an administrator, the user is able perform the following operations on the door lock:

1. Configure the door opening master key for a specific door lock - a master key can be requested to the web server with an administrator's Android application. In case a master key already exists in the web server's database, it is overwritten. After obtaining the new master key, the administrator may configure the door lock with a simple NFC interaction. In this case, along with the computed door lock challenge and the user-specific bytes, the NDEF message carries the updated master key and the current date and hour to configure the target door lock.

2. Administratively lock the door lock - this operation allows an administrator to deny access to any user without administrative privileges. Any door lock administrators are also able to unlock the door locks in this mode.
3. Reset the door lock - this operation resets the door lock, erasing its master key and current time. In this case, the door lock system is set to a zero-day mode, in which it may be configured by an administrator with a new time and master key and it only provides access to administrators, since all operations performed by administrators use maintenance key-based credentials.
4. Log retrieval - this operation aims to retrieve logs from a door lock, all past general use and administrative operations are logged. The log entries are stored in a rotative buffer in the door lock.

As previously stated, administrative operations require administrator authentication using a daily key which was derived using a maintenance key. These operations are identified by a single byte field within the NDEF message. The message's length varies according to the operation: when configuring a door lock a 4 bytes long field for user info, a 64 byte long field for the authentication code, a single byte field for operation code, a 16 bytes long field for the new master key and 6 bytes long field for the current time - and in all other cases, messages are the same length as the general use protocol messages (in this case, the master key and current time fields are excluded).

4.5 Brief performance overview

No detailed performance tests were made on the course of this work. Operations involved in the authentication process involve generating a random number (door lock side), sending NDEF messages, computing an HMAC value (mobile application), hashing the user ID (door lock) and performing an HDKF expansion (door lock). It can be assumed that the most CPU-intensive operations will be cryptographic computations in this case (HKDF expansion and HMAC calculation). As such, one can perform a theoretical estimative of the system's partial delay by analyzing the tabled time values [6] for hashing functions on an Arduino with an ARM processor, such as the one used in this work (M0). Android (the NDEF message sending should be considered as little significant, since only a few dozens of bytes are exchanged from each side). Hashing and finalization operations in HMAC computation using SHA256 corresponds to an estimated time of 240 microseconds (considering an ARM processor). In the case of a simple SHA256 operation to compute the user hash, the time is approximatively 78 microseconds. The total time will correspond to these two estimations added to the HMAC computation time on an Android device and the time of generating a random number. Therefore, it is possible to assume that time is not a decisive factor when evaluating a functional prototype of this system.

5 Security analysis

A critical aspect of this system is the secrecy of the doors' master keys, which cannot be easily disclosed. These keys are kept both by the authentication server

and by the doors, and are used to produce daily keys for the mobile application that interacts with the doors. This strategy resembles the Kerberos [10] paradigm: master keys are similar to service keys and daily keys are similar to session keys. The difference is that while in Kerberos the session keys are random, and need to be conveyed to services within tickets, our daily keys can be computed by the doors, thus they do not need to be conveyed to doors.

Another decisive aspect of the system, as the title of this article refers, is the possibility of off-line authentication, which adds another security layer to role-based access control systems since the access controller systems (in this case, the door locks) are not exposed to any network and therefore remote attacks or scans become harder or even impossible to explore.

Daily keys are derived from master keys, therefore we need to evaluate the protection of master keys from the analysis of daily keys. There are many key derivation processes, and many cryptographic protocols use them. In our case, we used the HKDF approach [5, 4], which is used in many protocols to create the so called key hierarchies.

The authentication of a door is simple while intuitive and effective. The door lock and operation selection process prevents malicious door openings in a passive type of interaction (which could imply several vulnerabilities, such as promptness to relay attacks), as it not possible to perform door lock and operation specific requests authentication request messages are always specific to a door lock and a service. But is it enough for preventing a user from interacting with a rogue door and provide a response for other door, which could then be used by the attacker? We believe this is not possible, since the identification of the door is provided by the door itself and should make sense for the user. In other words, if the user expects to see the identity X for the door with which they want to interact with and they get identity Y, then this is a clear evidence that someone is trying to get a valid response for a latter authentication in door Y. Therefore, we think that the door authentication paradigm is simple, though powerful enough to prevent dialogs with attackers impersonating doors.

6 Conclusions and future work

In this paper we have presented and NFC-enabled, off-line door lock system. The system allows an NFC-enabled device, belonging to a user, to request a particular service from the door, namely the opening of its lock. The authentication and authorization protocol involving the door and the mobile device does not require any dialog with a central authority during the protocol execution. Furthermore, all management operations involving the each door are conducted locally, also through NFC, using maintenance credentials and invoking a maintenance interaction with the door.

The system uses one pair of master keys per door, one for opening the door lock, the other for performing maintenance operations. Both keys are totally random, and shared between the door and a central authentication server. The

Integrating paper-based voting and Belenios
a hybrid voting protocol for an academic organization

Fernando Marques, Ana Almeida Matos, and Jan Cederquist

Instituto Superior Técnico
{fernando.m.marques,ana.matos,jan.cederquist}@tecnico.ulisboa.pt

Abstract. We consider the problem of introducing an electronic voting system in the context of an academic organization. We work under the assumption that such a context fits a profile where integrity of an election is expected to be fully verifiable, privacy must be ensured by the system, and coercion is not of primary concern. Additionally, we assume that a classic paper-voting is already in place, and while the voting community may be expected to be literate in handling web-based applications and in understanding the concept of verifiability, it may exhibit some cultural resistance against shifting to a new voting practice.
We present an integration of a classical paper ballot system with Belenios, an on-line voting system which guarantees vote privacy and verifiability. To this end, we propose a specification of a hybrid version of the Belenios protocol that integrates a classical paper ballot protocol. We then enunciate, up to assumptions of security of the two baseline protocols, the relevant security properties of the resulting protocol. Finally, we show that the new protocol is well suited to be adopted by an academic organization by presenting an architecture for the proposed solution that fits the current requirements of an existing university.

1 Introduction

Voting has been an important step in decision making for millenia. It has been used in different contexts, ranging from statewide elections and referendums, to those organized at the level of smaller societies [1–3]. Until recently, paper ballot systems have provided the classical medium for expressing votes, but the use of electronic voting (e-voting) has been gaining popularity over the last decades and is even currently being implemented in some countries for government elections.

E-voting may offer advantages such as closer geographical accessibility to the voter, faster publication of the results, and cost minimization of elections. More interestingly, it can equip the election process with novel security and auditability properties that are not achievable using paper voting. There is, however, a technical trade-off when managing simultaneously integrity and privacy of electoral data. Furthermore, there may exist trust issues among the voting population regarding the system where the election is being run, while people with less technical proficiency may find it hard to cast their vote due to not understanding how to work with the system. Indeed, the replacement of a classic paper voting system for an electronic one requires a cultural shift which is not easy.

A wide variety of electronic voting protocols have been proposed with the aim of improving the orchestration of the voting process. Each voting protocol may provide a set of guarantees under certain assumptions, and its suitability for a given election depends on how well the specific context of the election grants those assumptions and what requirements are considered of importance.

Problem. In this paper, we consider the problem of introducing an electronic voting system in the context of an academic organization. We work under the assumption that such a context fits a profile where integrity of an election is expected to be fully verifiable, where privacy must be ensured by the system, but where coercion is not a primary concern. Additionally, we assume that a classic paper-voting system is already in place, and while the voting community may be expected to be literate in handling web-based applications and in understanding the concept of verifiability, it may exhibit some cultural resistance against shifting to a new voting practice.

At an implementation level, we assume that an organization-wide authentication system is already in place, possibly associated to an integrated campus management system. Also, as available computing resources are often restricted, we focus on architectures that require a limited number of servers.

As a running example, we consider the concrete case of Instituto Superior Técnico (IST) of the University of Lisbon, where elections are run using a classical paper voting system. In this environment we have access to an authentication platform provided by IST informatics campus management system Fenix, and are limited to two servers. We expect that the results are applicable to similar organizations.

The main requirements for our voting system are thus:

- To depart as little as possible from the functionality and assurances of the paper ballot system which is already in place.
- To offer the functionality of electronic voting, associated to the desired properties of privacy and full integrity verifiability.
- To make use of the already existing authentication mechanisms for ease of usability.
- To be securely implementable using a low number of servers.

Solution. As already mentioned, beyond any technical arguments towards adopting an e-voting system, the replacement of a classic paper voting system for an electronic one requires a cultural shift. In order to mitigate the impact of this shift, the transition can be made by simultaneously enabling both electronic and paper-based voting methods.

We present an integration of a classical paper ballot system with the Belenios e-voting protocol, an on-line voting system which guarantees vote privacy and verifiability. To this end, we propose a specification of a hybrid version of the Belenios protocol which integrates the classical paper ballots. We then enunciate, up to assumptions of security of the two baseline protocols (Belenios and paper voting), the relevant security properties of the resulting protocol.

Belenios [4, 5] was chosen as a base protocol since it provides an easy to use voting system that guarantees vote privacy and end-to-end-verifiability. It is based on a homomorphic encryption scheme [6] with additive properties and a credential scheme which allows to identify a vote to a voter. It retains a simple specification since the protocol does not require a large number of servers.

The main security properties and assumptions that hold for our system are:

Integrity Ballots are counted correctly, independently of whether they were cast in paper and/or electronically. This implies in particular consistency between the e-voting and paper voting. Holds under assumptions of correct (human) handling and tallying of the paper votes, and of non-corruptability of both the credential authority and the voting server simultaneously.

Confidentiality The association between the object of a vote and the voter is known only by the voter, up to aggregation of results. Holds under the assumption that the cryptographic scheme used to encrypt the ballots is not broken and that the trustees of the protocol don't work together with malicious intent.

Auditability The voter can verify that her electronic vote is registered correctly. Any user can verify that the counting of the registered electronically cast votes is done correctly. Counting of votes cast by paper can be verified within the defined time frame before they are destroyed. Holds by design of the protocol, under the assumption that it is correctly implemented.

Contributions. The main technical contributions are the following:

- A new hybrid voting system that integrates a modified version of the Belenios e-voting protocol (in which voting credentials are generated on demand) and a classical paper voting protocol.
- Accurate enunciation of the preservation of properties of the baseline protocols.
- Proof of concept of applicability to an academic setting, by the design of an architecture for implementing the proposed solution that fits the implementation requirements of an existing organization.

Overview. We start by presenting, in Section 2, the two baseline protocols that are integrated into the proposed hybrid protocol, which we describe in detail. For each we present the assumptions and main security properties. In particular, we enunciate the relevant security properties that hold for the hybrid protocol. In Section 3, we present the architecture of the system. Finally, in the last two sections we present the related work, and conclude.

2 Protocols

In this section we present a new hybrid e-voting system which we refer to as *H-Belenios*. We start by summarizing the two baseline protocols that will be integrated into our hybrid protocol and their main security properties: *Baseline*

Protocol A (BP-A), refers to the paper protocol, and *Baseline Protocol B (BP-B)* refers to the e-voting protocol. We then describe and formally specify the H-Belenios protocol.

In all three protocols, we distinguish the following four main phases: The *Set up phase* includes the steps that are necessary to specify the election parties, and to generate and input the necessary configurations for the election to occur; the *Voting phase*, includes the processes by which voters may cast their votes; the *Tally phase*, when all the votes are added and the final result of the election is published; and the *Audit phase*, which may overlap with the voting phase, when (partial) verification of correctness of (partial) results can be performed.

We refer to a user who is authorized to vote in a specific election as an *eligible voter*. Furthermore, the *final vote of an eligible voter* is either the paper vote (if the voter has cast a paper vote), the vote cast electronically (if the voter did not cast a paper vote), or undefined (if the voter did not vote).

2.1 Baseline Protocol A: Paper voting

The following protocol, which we refer to as Baseline Protocol A (BP-A), is based on the regulations for the establishment of organs within IST [7].

The main parties involved in BP-A are the electoral commission, the members present at each table which are the poll workers, and each eligible voter.

Set up phase. Before the election begins, the members of the electoral commission, including the president, are designated. These members cannot be candidates for the election and include, in particular, one member per election candidate which serves as a representative.
On the day of the election, one or more voting tables are available for depositing votes. Each voting table has a group of pool workers (which may include one representative per candidate) designated to it.

Voting phase. In order to vote, the voter first shows his identity to the voting table president, who also verifies his voting right. The voting secretary notes on the electoral roll that this voter has voted and the president hands out a paper ballot. The voter goes to a voting booth, and fills in the ballot secretly by placing an 'X' next to the candidate of choice. Any other type of mark on the ballot invalidates it. Finally, the voter hands in the folded ballot to a person present at the voting table who inserts it into the ballot box.

Tally phase. At the end of the election the pool workers at each table proceed to count the votes and their distribution. The minutes are then written, recording the information gathered from the votes. The partial results are then handed to the electoral commission who will add all the votes from different tables and calculate the tally. The results must be released 24 hours after the end of the voting phase, and the paper votes must be destroyed 30 days after the election has passed.

Audit phase. After the tally phase complaints can be made to the electoral commission with a time limit of one day after the results are published. Every pool worker also has the option of complaining in the table report against decisions made by that table.

Assumptions:

- No one, including the pool workers, interferes with the paper votes in the ballot boxes.
- The pool workers perform the tally and publication of the results according to regulations.

Security properties:

- Only and all eligible voters are able to vote.
- Each vote is kept confidential (regarding both existence of vote and vote content, for a given voter), up to aggregation of results.
- The regulations specify a correct procedure for counting exactly one vote per voter.
- It is possible to retally the votes, within a specified time frame.

2.2 Baseline Protocol B: Belenios

The following protocol, which we refer to as Baseline Protocol B (BP-B), summarizes the core of the Belenios electronic election protocol (see [5] for the full formal specification) on which we base the electronic part of our voting system. Other features, such as the possibility of credential recovery, are omitted.

The main parties involved in BP-B are the voting server, the server administrator, the credential authority, each eligible voter and each trustee.

Set up phase. An election in Belenios starts with the creation of the credentials. The server administrator sends a unique universal identifier (uuid) to the credential authority. There, a list of credentials and their public parts is created. Each credential is sent to the respective voter, and the list of shuffled public parts for the credentials is generated and sent to the server administrator.

Then, each trustee generates her own public key and sends it to the server administrator, who verifies that the trustee has the secret part of it, and in the end combines all of the trustees keys in order to generate the election public key. With both the list of public credentials and the election public key, the server administrator is able to create the election. The protocol uses a public key encryption scheme with additive homomorphic properties.

Voting phase. In order to vote, the voter obtains the election parameters, creates a ballot which is formed from the encrypted votes and proofs (proof of membership and proof that the vote is formed correctly), the signature and the election uuid, and obtains a hash of the ballot. If all the data present in the ballot is valid then the voting server publishes it.

Tally phase. The election is terminated by the server administrator. The homomorphic properties of the encryption scheme are used to aggregate all the votes into an encrypted tally which is sent to each trustee to perform a partial decryption. Finally, the server administrator verifies the partial decryptions and aggregates them in order to create the decrypted tally, which is then published.

Audit phase. During the voting phase and tally phase, each voter is able to verify that their vote was cast correctly using the smart ballot tracker (hash of the serialization of the ballot) which was published on the bulletin board when the ballot was cast. It is also possible to verify that the encrypted tally was calculated correctly. The greater the number of voters performing this verification, then stronger is the confidence in the outcome of the election.

Assumptions:

- Not both the voting server and the credential authority are simultaneously corrupt.
- The client software does not leak information about the electronic votes.
- The trustees will not work together in a malicious way.

Security properties:

- Only and all eligible voters are able to vote.
- Each vote is kept confidential (regarding vote content), up to aggregation of results.
- The protocol specifies a correct procedure for counting exactly one vote per voter.
- Each voter is able to verify that their vote was cast correctly.
- Each voter is able to verify that the tally is correct.

2.3 Hybrid Protocol: H-Belenios

The parties that participate in the hybrid protocol include those of BP-B – the voting server (**VS**), the credential authority (**CA**), the server administrator (**SA**), each eligible voter (**V**) and each trustee (**T**) –, as well as that which is present in the paper part of the protocol – the poll worker (**PW**).

In order to formally describe the integration of the two baseline protocols, we make explicit the different structures where data is saved: The **EL**, which is a list of all the elections, including their parameters (dates, descriptions, names) and their uuid; the eligible voter list **EVL**, which maps each voter identifier to their respective public credential; the list **TL** of trustees; the table of electronic votes **TEV** which contain all the e-votes done by eligible voters; the list of identifiers of paper voters **LPV**, that is the representation of the paper voters on the system and the table of final electronic votes **TFEV**. We also have **pLPV** which is the same as **LPV** but only in paper format, as is currently used in BP-A. From these structures, during the voting phase the smart ballot tracker of each ballot in **TEV** will be published and after the tally the smart ballot tracker of each ballot in **TFEV** will be made available together with the voter identifiers of those who voted by paper. The data structures which are made public will be available to view in the smart bulletin board **SBB**.

In the following description of H-Belenios, in order to avoid redundancy, we focus on the differences with respect to the baseline protocols. For each phase of the election, a table presents the corresponding formal description. We also assume secure channels are being used for message exchange.

Set up phase. As in BP-B, to create the electronic counterpart of an election, we require the basic parameters of the election (name, dates and other information, denoted by eParam), the list of eligible voters with their voter identifiers **EVL** and the list of trustees **TL**, the election public key Pk_e which is made from the trustee's ($t \in \mathbf{TL}$) public key shares and proofs that they each have the secret key share δ_t, and the election identifier which is an uuid e_{id} (which is sent in all message exchanges throughout the protocol). These cryptographic keys, which are needed for electronic voting, are generated in a similar way as in BP-B. However, in H-Belenios, they are generated on demand for voters who wish to vote electronically.

#	Message Exchange	Voting Server & Administrator
1	$\mathbf{SA} \to \mathbf{VS}$: eParam	
2		$\mathbf{EL} \coloneqq \mathbf{EL} \cup \{\langle e_{id}, \text{eParam}\rangle\}$
3		publish($\langle e_{id}, \text{eParam}\rangle$)
4	$\mathbf{SA} \to \mathbf{VS} : \mathbf{TL}$	
5	$t \to \mathbf{VS} : Pk_t, \delta_t, \forall t \in \mathbf{TL}$	
6		publish(Pk_t), $\forall t \in \mathbf{TL}$
7	$\mathbf{VS} \to \mathbf{SA} : Pk_t, \delta_t, \forall t \in \mathbf{TL}$	
8		validate(δ_t), $\forall t \in \mathbf{TL}$
9	$\mathbf{SA} \to \mathbf{VS} : Pk_e = \sum_{t \in \mathbf{TL}} Pk_t$	
10		publish(Pk_e)
11	$\mathbf{SA} \to \mathbf{VS} : \mathbf{EVL}$	

Voting phase. The voting procedure can be initiated by any voter $v \in \mathbf{EVL}$. In order to vote electronically, v must first request his credentials. This will not prevent v from also voting by paper (which would take precedence over the electronic vote). The voter will then receive by email from **CA** the public and secret credentials, respectively denoted by Pc and Sc, that are necessary to vote. The connection between the Pc and the voter's Id is maintained in **EVL** in order to allow revocation of electronic votes due to casting of paper votes. In the following, p, q denote large prime numbers, and g the generator for the cyclic group of order p, which are used for the encryption and signature schemes that allow to confidentially send electronic ballots (eBallot) carrying the encrypted answers (ans) to the question of the election.

#	Message Exchange	Voting Server
1	$v \to \mathbf{VS} : Id_v$, request	
2	$\mathbf{VS} \to \mathbf{CA} : Id_v, p, q, g$	
3	$\mathbf{CA} \to \mathbf{VS} : Pc_v, Id_v$	
4		$\mathbf{EVL} \coloneqq \mathbf{EVL}[Id_v \mapsto Pc_v]$
5	$\mathbf{CA} \to v : Pc_v, Sc_v$	
6	$\mathbf{VS} \to v : \text{eParam}, p, q, g, Pk_e$	
7	$v \to \mathbf{VS} : \text{eBallot} = ans, s\{ans\}_{Sc_v}, \delta_{ans}, Pc_v$	
8		validate($Pc_v \notin \mathbf{TEV}$)
9		$\mathbf{TEV} \coloneqq \mathbf{TEV} \cup \{\langle Pc_v, \text{eBallot}\rangle\}$
10		publish(**TEV**)

For the sake of consistency with BP-B, we allow a single electronic vote per voter. However, there is no technical obstacle to generalising this phase in

order to allow any number of electronic votes, whilst only the latest one would be considered (without affecting the precedence of a paper vote over it).
The steps for voting by paper are the same as in BP-A. The only difference is that at the end of this phase, the list of paper voters of the voting tables **pLPV** must be delivered to **SA** who must form **LPV** and input it to **VS**. We denote by **BB** the set of paper votes cast in the ballot box, and each paper ballot is denoted as pBallot.

#	Message Exchange	Server & Administrator	Voting Table
1	$v \rightarrow \mathbf{PW} : Id_v$		
2			validate($Id_v \in \text{dom}(\mathbf{EVL}) \setminus \mathbf{pLPV}$)
3	$v \rightarrow \mathbf{PW}$: pBallot		
4		$\mathbf{pLPV} \coloneqq \mathbf{pLPV} \cup \{Id_v\}$	
5			$\mathbf{BB} \coloneqq \mathbf{BB} \cup \{\text{pBallot}_{id}\}$
6	$\mathbf{PW} \rightarrow \mathbf{SA} : \mathbf{pLPV}$		
7		$\mathbf{LPV} \coloneqq \mathbf{pLPV}$	
8	$\mathbf{SA} \rightarrow \mathbf{VS} : \mathbf{LPV}$		

Tally phase. The **VS** aggregates all the final electronic votes using the homomorphic properties of the encryption scheme. The electronic votes from the voters who also voted by paper are omitted in **TFEV**. These votes are identified by using the connection between their voter *Id* and the *Pc* which is present in the electronic ballot. This partial encrypted tally (denoted bellow by eRes) is sent to each trustee ($t \in \mathbf{TL}$) who computes a partial decryption ($D_t\{\text{eRes}\}$) and a proof of correct decryption. The **SA** in the end verifies all decryptions, and aggregates them ($D\{\text{eRes}\}$). At this time the **SA** must input the paper results to the server. After the tally and the verifications are done, then the results can be posted on the bulletin board.

#	Message Exchange	Server & Administrator	Voting Table
1		$\mathbf{TFEV} \coloneqq \mathbf{TEV}\vert_{\overline{\mathbf{EVL(LPV)}}}$	
2		eRes $\coloneqq$ tally($\mathbf{TFEV}$)	
3			pRes $\coloneqq$ tally($\mathbf{BB}$)
4	$\mathbf{VS} \rightarrow t$: eRes, $\forall t \in \mathbf{TL}$		
5	$x \rightarrow \mathbf{VS} : D_t\{\text{eRes}\}, \delta_{D_t}, \forall t \in \mathbf{TL}$		
6	$\mathbf{VS} \rightarrow \mathbf{SA} : D_t\{\text{eRes}\}, \delta_{D_t}, \forall t \in \mathbf{TL}$		
7		validate(δ_{D_t}), $\forall t \in \mathbf{TL}$	
8	$\mathbf{SA} \rightarrow \mathbf{VS} : D\{\text{eRes}\}$		
9	$\mathbf{SA} \rightarrow \mathbf{VS}$: pRes		
10		fRes $\coloneqq D\{\text{eRes}\} +$ pRes	
11		publish($\mathbf{TFEV}, \mathbf{LPV}$)	
12		publish(fRes)	

Audit phase. The auditing of the paper and the electronic part of the voting process will occur in a similar fashion as for BP-A and BP-B, respectively. The main difference is that at the end of the election, only the smart ballot trackers of the final electronic votes are published. For those voters who voted by paper, their identifier will appear saying that they voted as such. Furthermore, we add a checksum that validates de total number of votes

against those that appear in the smart ballot tracker and the votes done by paper (given by $|\text{finalResult}| == |\mathbf{TFEV}| + |\mathbf{LPV}|$). This enables to detect any mismatch between the number of counted paper votes and those that appear in the smart ballot tracker.

Assumptions: The union of those enunciated for the baseline protocols A and B.

Security properties:

- Only and all eligible voters are able to vote.
- Each vote is kept confidential (regarding vote content), up to aggregation of results.
- The protocol specifies a correct procedure for counting exactly one vote per voter.
- It is possible to retally the paper votes, within a specified time frame.
- Assuming that the tally of the paper votes is correct, every voter is able to verify that the final tally is correct.

The proofs of the above properties rely on the presented properties of the baseline protocols, and on the specified steps of the H-Belenios protocol.

Coercion resistance. While BP-A is considered to be strongly coercion resistant, BP-B suffers from the fact that vote receipts can be produced. Although this is assumed to not be of primary concern in the context of this work, we observe that as a result of the integration of the protocol with a paper-based voting system, the new protocol improves on the level of coercion resistance that is provided by BP-B. Indeed, the voter now has a choice of also casting a paper ballot, which is preferred over the electronic ballot during the tally. As such there is no way to prove who he voted for, up to aggregation of the results.

Relaxing assumptions. The integrity properties of BP-A rely on strong assumptions of correctness of the actions of the poll workers. However, these assumptions can be relaxed in face of the new possibility of detecting a mismatch between the counted number of paper votes and the electronic votes published in the smart bulletin board. More precisely, it is no longer necessary to assume that fake ballots or votes cannot be added to or removed from the ballot boxes or tally, but only that they are not changed.

3 Architecture

As described in Figure 1, the system will have two servers running, the voting server, which will work as a bulletin board and ballot box, and the credential authority, which will be responsible for the generation and distribution of the elections. The connections between every component and entity will be done using TLS connections in order to avoid man in the middle attacks. The entities will connect to the voting server using a web browser, that will run a web application, and they will need to authenticate themselves using the Fenix platform authentication credentials, which is possible due to the use of Oauth2.

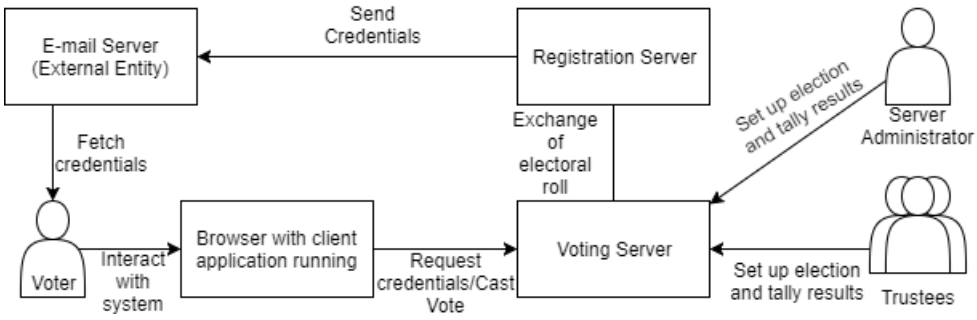


Fig. 1: Architecture of the protocol

4 Related Work

Internet voting protocols. We base the electronic part of our hybrid e-voting protocol on Belenios. Belenios is a purely e-voting protocol which is in turn based on Helios [8] and Helios-C [9]. Helios was created by Ben Adida in order to allow voters to audit their ballots before casting them. Various versions of Helios have been released, incorporating new security features such as the replacement of mix-networks [10], used in the anonymization of the ballots, for the use of homomorphic encryption using the El Gamal encryption scheme [11]. One of the main differences between Helios protocol and Belenios is that the later uses a credential system in order to avoid ballot stuffing. Helios-C is an improvement over the Helios system, proposed by Veronique Cortier et al., which adds the idea of credentials. This protocol uses an El Gamal encryption scheme with homomorphic properties for anonymization and a Schnorr signature scheme [12] for ballot authenticity. The main difference between Belenios and Helios-C is that the former does not have threshold support for key generation.

Hybrid voting protocols. TrustVote [13] is a hybrid e-voting system which includes a paper-based and an electronic-based voting components, designed to be used by governments and organizations. The e-voting component is based on FOO92 [14], being one of the main improvements the introduction of threshold blind signatures. This system also allows the possibility of revoking a previous electronic vote. In order to integrate the paper voting with their e-voting system, paper voting is allowed during a new period that starts after the electronic vote casting ended. If the voter had cast an electronic vote, then during the paper voting he may chose to revoke it. In order to revoke an electronic vote, the voter must present his voter secret which is used in order to identify his e-vote. The tally is then computed with the sum of all electronic votes minus the sum of all revoked electronic votes and finally adding the sum of all paper votes. Unlike our system, paper voting and e-voting phases are thus kept separate in TrustVote. It is also necessary to submit additional documents in the act of paper voting.

Another implementation of a hybrid e-voting system was proposed by Oliver Spycher and Rolf Haenni [15]. This system makes use of pseudonyms created

from the credentials given to the voters. These pseudonyms are shuffled with the public credentials. The voter can then cast an electronic ballot with the pseudonym created from the credentials, the encrypted vote and zero knowledge proof, which are published on the public bulletin board. If a voter chooses to vote by paper he must present at the polling station his public credential, his pseudonym and a zero knowledge proof that verifies that he gave the right pseudonym. If there is a vote associated with the pseudonym, then that vote must be revoked and the voter may then vote by paper; otherwise the voter can vote by paper since he has proven with the credentials, pseudonym and zero knowledge proof that he is an eligible voter. Similarly to TrustVote, this protocol also needs the submission of the voter's public credential.

Voting in academic organizations. Many academic organizations still rely exclusively on paper voting, but there are some who have started to adopt electronic voting systems. Some examples of such universities are Princeton who currently have a Helios server running for elections [16], Université Catholique de Louvain (UCL) in Louvain-la-Neuve, Belgium, which also uses a version of Helios [17] and MIT who used EVOX for its Undergraduate Association elections [18].

5 Conclusions and Future Work

This paper presents a hybrid voting protocol that integrates a well-studied e-voting protocol with a standard paper-based protocol. By minimizing the changes to the later, we aim to mitigate the cultural shift that is required in a transition towards a full e-voting system.

We consider the concrete case of an existing academic organization (IST) as a proof-of-concept that the proposed solution satisfies the enunciated requirements. Furthermore, the solution is currently being developed for coupling with IST's integrated campus management system (Fenix). We expect that other academic organizations have similar requirements and computing contexts (eg., an internal authentication system), and that therefore our proposed protocol is useful beyond the specific case of IST.

As future work, we envision to extend the protocol with full receipt freeness, as proposed and previously discussed for BeleniosRF. Another possible improvement would be to adapt the protocol in order to distribute the credentials in more secure ways, as for instance by means of smart cards. There is also the possibility of having a centralized list of voters so that each voter can vote by paper in any of the tables which are available.

Besides offering an intermediate step towards the change from a exclusive paper voting community to one that embraces with a bit more ease the idea of e-voting, new advantages entail from preserving the possibility of paper voting: On one hand, partial coercion freeness is added to the electronic component of the protocol, resulting simply from the possibility of overriding any electronic vote by means of a paper vote. On another hand, prevention of ballot stuffing and elimination in the paper-based protocol, which relied solely on assumptions

Geração Automática de Conhecimento para SDI extraído de OSINTs

Ivo Vacas, Ibéria Medeiros

LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal.
ivo.ricardo1@gmail.com,imedeiros@di.fc.ul.pt

Resumo O cibercrime a organizações tem sido uma constante nos dias de hoje. As organizações para se protegerem destes ataques utilizam mecanismos de defesa, tais como sistemas detetores de intrusões (SDI), no entanto, a eficácia dos SDIs na deteção destes ataques depende do conhecimento que estes contêm sobre as ameaças e da forma como as detetam. O contínuo aparecimento de novos e sofisticados ataques torna obrigatório que os SDIs sejam atualizados constantemente com conhecimento sobre novas ameaças. Este conhecimento pode ser obtido de diversas fontes de inteligência - *Open Source Intelligence* (OSINT) - públicas, que se encontram acessíveis em diversos locais na Internet. Este artigo apresenta uma solução para melhorar uma arquitetura de deteção de intrusões em SDI. A solução propõe um *gerador de regras e blacklists* para SDI, com base em informação OSINT recolhida por uma plataforma de *threat intelligence* bem como a sua integração no SDI de forma automática. Foi realizada uma avaliação experimental da solução em ambiente real, usando 49 fontes de OSINT coletadas pela plataforma *threat intelligence* IntelMQ e o SDI Snort. A arquitetura proposta permitiu detetar ameaças de diversas categorias.

1 Introdução

O cibercrime a organizações tem sido uma constante nos dias de hoje. Ataques a servidores organizacionais, tais como o *Distributed Denial of Service* (DDoS), *spamming* e propagação de *malware*, são cada vez mais frequentes, de tal forma que prevê-se que em 2019 o custo dos seus danos atinja 2 triliões de dólares [9]. Para a concretização destes ataques, geralmente é previamente necessário infetar as máquinas vítimas com *malware* ou *backdoors* para posteriormente os ciber-criminosos remotamente as poderem controlar e realizar os ataques [1]. As máquinas mais apetecíveis de infetar são as com acesso à Internet, pouca monitorização e grande largura de banda [3]. Quando comprometidas são chamadas de *bots* e as redes por elas formadas de *botnets*.

A utilização de *botnets* tem sido um dos recursos pertencentes aos vetores de ataque do cibercrime, em tal escala, que de acordo com o relatório de Akamai, estes ataques aumentaram em 9% do primeiro para o segundo quadrimestre de 2016 e em 129% comparativamente com o ano anterior [33]. Por exemplo, a Mirai *botnet* esteve na base deste aumento. Esta *botnet* é caracterizada pelos seus *bots* serem dispositivos IoT (*Internet of Things*), e em outubro de 2016 foi usada para atacar os servidores da

Dyn (organização que controla servidores de DNS da Internet), onde participaram 100.000 dispositivos IoT que geraram um volume de tráfego na ordem dos 1.2 Tbps, de tal forma que provocou uma interrupção do serviço de Internet em toda a Europa e EUA [4]. Neste ataque poderiam ter participado mais de 500.000 dispositivos IoT, por estarem vulneráveis à Mirai [29].

Face a este problema, as organizações para se protegerem utilizam sistemas detetores de intrusões (SDI) que monitorizam o tráfego da rede, detetando anomalias e gerando alertas destas. O modo primordial de funcionamento dos SDI é baseado no conhecimento do comportamento ou de assinaturas, que quando há um desvio de comportamento ou igualdade de assinatura, o SDI gera um alerta [7]. No entanto, estes sistemas são conhecidos por reportarem falsos positivos (alertas de falsas anomalias) devido a ligeiros desvios comportamentais que possam ocorrer, ao contrário dos falsos negativos (anomalias não detetadas) devido a não possuírem o conhecimento suficiente sobre todas as anomalias e ataques existentes [10].

Apesar da importância dos SDIs na deteção de ameaças, nomeadamente naqueles que utilizam *botnets*, existe a necessidade constante de fornecer conhecimento aos SDIs, pois novos e sofisticados ataques surgem muito rapidamente, deixando os SDIs desatualizados. A obtenção deste conhecimento, por um lado, é considerada privada e segredo de negócio [8], não estando portanto acessível a todos. Por seu turno, este conhecimento pode ser obtido de diversas fontes de inteligência – *Open Source Intelligence* (OSINT) – públicas e acessíveis em diversos locais na Internet. O OSINT contem informação partilhada proveniente de *honeypots* e de eventos reais que causaram algum tipo de dano em dados ou serviços, como o roubo de credenciais por ataques de *Phishing* ou DDoS [7]. Por vezes, esta informação por si só é reduzida e insuficiente para caracterizar ameaças em concreto e ser aplicada na sua detecção, no entanto, se agregada e complementada com outra informação é possível criar IOA (*Indicators of Attack*) [32] que permitem caracterizar com clareza ataques e serem usados na sua deteção.

Este artigo apresenta uma solução para melhorar uma arquitetura de deteção de intrusões em SDI. A solução propõe um *gerador automático de regras e blacklists* para o SDI, com base em IOAs gerados a partir da extração de conhecimento OSINT por uma plataforma de *threat intelligence*, bem como a sua completude e integração no SDI de forma automática. A solução foi implementada e avaliada num ambiente real, na Reitoria da Universidade de Lisboa, usando 49 fontes de OSINT coletadas pela plataforma *threat intelligence* IntelMQ, que após de filtradas e reunida informação necessária e suficiente para criar IOAs, o gerador automaticamente gera regras e *blacklists*, os quais são integrados de forma automática no SDI Snort. A solução proposta permitiu detetar ameaças de diversas categorias.

As contribuições deste trabalho são: 1) uma arquitetura para extração de conhecimento OSINT e criação de IOAs; 2) um gerador de regras e de *blacklists* para SDI a partir de IOAs; 3) uma implementação da solução de extração de OSINT e geração de regras e *blacklists*; 4) uma avaliação experimental da solução em ambiente real.

2 Botnets, ataques e SDIs

Nesta seção é apesentado brevemente como são compostas as *botnets* e ataques com elas realizadas, e as características principais de um sistema detetor de intrusões.

As *botnets* são redes de dispositivos vulneráveis que foram infetados, tais como computadores e dispositivos móveis, e controladas por entidades criminosas (*botmaster*) através de um centro de comando e controlo (C&C) [30]. O uso de *botnets* permite que essas entidades realizem ataques usando recursos que não lhes pertencem, mantendo-se assim anónimas [2]. Um dispositivo infetado (ex., pela instalação de *malware*) e controlado por um *botmaster* é apelidado de *bot*. Os *bots* recebem comandos do *botmaster* para realizarem ações ilícitas, mantendo o anonimato do seu dono.

O C&C das *botnets* é um alvo particularmente apetecível, quer para quem as tenta descobrir e destruir, quer para outros ciber-criminosos que pretendem apropriar-se delas para poderem usa-las em seu proveito. Os donos das *botnets* utilizam mecanismos criptográficos para protegerem a comunicação nas *botnets* e mecanismos de dissimulação para impedirem a sua identificação e desmantelamento [6], evitando assim a sua captura por sistemas de monitorização de redes de computadores, tais como os SDIs. Por exemplo, um mecanismo eficaz de dissimulação de *bots* é o de metamorfose, que permite que estes sejam difíceis de detetar pelos SDIs devido à sua constante mudança comportamental [5].

As *botnets* têm estado na base de diversos tipos de ataques, sendo o *SPAM*, propagação de *malware*, *ransomware*, *phishing*, e DDoS os ataques mais realizados. Existe uma grande dificuldade em eliminar estes ataques devido à constante sofisticação dos métodos de operação dos atacantes e do constante mascaramento das *botnets*.

Os SDI têm contribuído em muito na deteção destes ataques. Conseguem analisar o tráfego gerado nas instituições e reconhecer ameaças através de padrões sobre estas (ex., ataques). Um SDI é tão eficaz quanto o conhecimento que possui das ameaças. Os SDIs detetam intrusões *baseado em rede* e *baseado em host*. Os SDIs baseados em rede são caracterizados por analisarem tráfego de rede, enquanto os SDI baseado em *host* analisam um sistema (ex., computador). Ambas as formas de análise podem ser conseguidas pelos métodos baseados em *assinatura* ou *comportamento* [7].

Os SDI baseados em assinatura utilizam bases de dados de assinaturas de ameaças. Este método analisa os dados que chegam ao SDI, verificando se na base de dados existe uma assinatura que combine com a assinatura dos dados recebidos, denunciando assim uma intrusão. Apesar da trivial implementação de sistemas deste tipo, esta solução apenas permite detetar intrusões conhecidas e que constem na base de dados. Este método, em regra, tem a vantagem de possuir uma taxa de falsos positivos baixa e fornecer informação sobre os alarmes despoletados. Contudo, a taxa de falsos negativos poderá ser elevada devido às intrusões desconhecidas pelo SDI [10] [7].

Os SDI baseados em comportamento é o método mais investigado na área de deteção de intrusões. Tendo como primitiva o total conhecimento do correto funcionamento do sistema em que se insere, este tipo de SDI consegue detetar padrões comportamentais considerados anormais e gerar alarmes relativos a esses desvios comportamentais. Estes tipos de sistemas são complexos e envolvem um estudo extensivo e

constante do sistema a ser analisado. Contrariamente aos SDI's baseados em assinaturas, este método, em regra, tem uma taxa de falsos negativos muito baixa, conseguindo detetar intrusões com comportamentos desconhecidos e divergentes, como por exemplo, exploração de vulnerabilidades. Contudo, esta solução, tende a ter uma taxa de falsos positivos elevada, pois por qualquer desvio do comportamento conhecido é gerado um alerta, no entanto, o comportamento registado pode não ser malicioso.

3 Visão Geral da Arquitetura

A arquitetura da solução proposta é composta por três componentes: *recolha de informação*, *geração de conhecimento* e *deteção de incidentes*. A primeira recolhe e agrega informação de eventos de segurança de *feeds* OSINT, enquanto a segunda, com base nesta informação, gera conhecimento para SDI sob a forma de regras e *blacklists*, e a terceira deteta incidentes pela aplicação do conhecimento gerado.

As três componentes estão representadas na Figura 1. A componente de recolha de informação executa os seguintes passos: o *OSINT colector* recolhe os eventos de segurança dos diversos feeds (fontes) de OSINT. Seguidamente o módulo *Threat information extractor* processa os eventos, sob a forma de IOC (*Indicator of Compromise*, fragmentos de informação forense que podem ser utilizados para identificar potenciais actividades maliciosas num sistema [31] [32]), extraindo a informação relevante sobre ameaças, para seguidamente ser agregada por tipo de ameaça pelo módulo *Threat information aggregator*. A componente de geração de conhecimento, através do módulo *Event generator*, recebe a informação agregada, adiciona-lhe dados relevantes, transformando os IOCs em IOAs (*Indicators of Attack*, contêm informação que permite identificar ataques em execução [31] [32]), e armazena os IOAs gerados. Por fim, o *Rules & blacklist generator* recebe os IOAs e gera regras para o SDI, no formato específico do SDI, e *blacklists* com IPs maliciosos. A componente de deteção de incidentes, o *Rules & blacklist manager* gerencia as novas regras e *blacklists*, vindas da componente geração de conhecimento e do exterior (*Rules* na figura), para de seguida fornecer tal conhecimento ao SDI para este detetar incidentes.

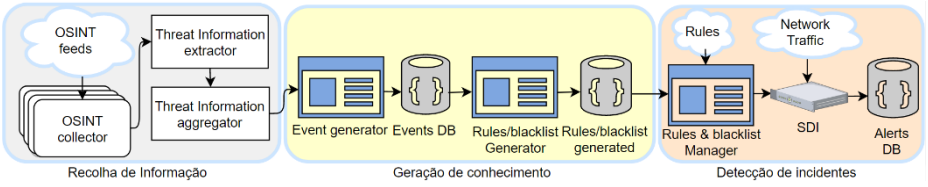


Figura 1. Arquitetura da solução proposta com os três componentes.

4 Implementação

A arquitetura proposta foi implementada usando a plataforma de *threat intelligence IntelMQ* para coletar eventos de 49 fontes OSINT de 8 categorias de eventos de segurança, extrair e agregar informação destes. Os módulos *event generator* e o *Rules &*

blacklists generator foram desenvolvidos por nós para, respetivamente, interagir com o IntelMQ e processar os seus resultados. A implementação foi obtida pelas fases de *gestão do conhecimento* e *deteção de incidentes*, como ilustra a Figura 2. A parte esquerda da figura, gestão do conhecimento, implementa as componentes de recolha de informação e geração de conhecimento da arquitetura proposta, enquanto a parte direita da figura implementa a componente de deteção de incidentes. A fase de gestão do conhecimento é executada diariamente para recolha e processamento de novos eventos de OSINT e geração de conhecimento para SDI, enquanto a fase de detecção de incidentes está em contínua execução. A seção apresenta de seguida os detalhes e funcionamento de cada um dos elementos pertencentes às duas fases.

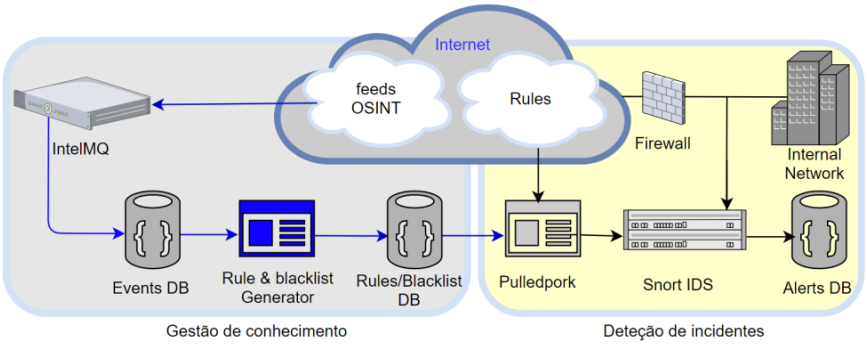


Figura 2. Implementação da arquitetura proposta, com as fases de gestão de conhecimento (esquerda) e deteção de incidentes (direita).

4.1 OSINT feeds

Com o objetivo de reduzir o cibercrime e salvaguardar a integridade intelectual, existem empresas, aficionados, investigadores e instituições (*feeds*) que disponibilizam informação à comunidade cibernética sobre incidentes de segurança (ex., sob a forma de IOC), tais como IPs maliciosos, sites de *phishing*, domínios perigosos, entre outras fontes relevantes. De acordo com a especialidade, alguns exemplos de *feeds* são o Phishtank [16] para *phishing* e CINSScore [17] para reputação de IPs. Esta informação se usada adequadamente e atempadamente pode prevenir a concretização de incidentes perigosos, tais como o roubo de credenciais. A arquitetura implementada foi configurada com 49 *feeds* de OSINT, a partir de 44 repositórios e que foram estratificados em 8 categorias de eventos de segurança. A Tabela 1 apresenta esta divisão pelas categorias de eventos OSINT (colunas 1 e 3), totalizando os 49 *feeds*.

Evento OSINT	**Feeds**	**Evento OSINT**	**Feeds**
Domínio Malicioso	9	IP's que atacam servidores de EMAIL	2
Blacklist IP	16	IP's que atacam serviços de VoIP	4
Phishing ou links perigosos	7	IP's que efetuam scans através de SNMP	1
IP's que atacam servidores FTP	1	IP's que tentam obter acesso remoto	9
Total	**33**	**Total**	**16**

Tabela 1. Categorização dos eventos de OSINT.

4.2 IntelMQ

A ferramenta IntelMQ foi desenhada pelo CERT (*Computer Emergency Response Team*) de alguns países Europeus com o objetivo de facilitar a recolha e tratamento de dados relativo a ameaças e melhorar a resposta a incidentes [11].

No IntelMQ, os *bots* são blocos de código modulares e independentes, que comunicam entre si sob a forma de mensagens em formato JSON, o que permite agilizar a comunicação entre eles. Existem diversos tipos de bots e para diversas funções.

O fluxo de dados do IntelMQ é apresentado na Figura 3. O fluxo inicia-se pelos *coletores*, que são os *bots* responsáveis pela recolha de informação de um dado *feed* OSINT e passagem desta aos *parsers*. Foram configurados 44 coletores OSINT de diversas entidades e respetivos *parsers*. O *parser* analisa a informação e cria eventos a partir desta que serão os *input* dos *experts*. O *deduplicator* remove os eventos duplicados, enquanto o *information expert* acrescenta informação relevante sobre o incidente, por exemplo o *source.asn* que representa o sistema autónomo de origem de um IP (à direita na figura). Os *experts* processam esta informação, obtendo mais informação ou despoletando novas ações, resultando os IOCs e eventos de *blacklist*. O *protocol expert* é um *bot* programado por nós para identificar o protocolo afeto ao IOC, com recurso a expressões regulares (*regex*), e preencher o campo *protocol.application* com esta informação (à direita da figura). Este campo é acrescentado aos IOCs, transformando-os assim em IOAs, e servirá para orientar o módulo *Rules & blacklists generator* na criação de regras e na decisão do tipo de regra a gerar. Desta forma, as regras criadas vão conter mais informação sobre o evento, minimizando assim a tendência de originar falsos positivos. Por fim, o *event writer* recebe os resultados do protocolo expert, formata-os segundo IOAs e armazena os IOAs em *events DB*. O *event writer* é um *output bot* que foi desenvolvido para este efeito.

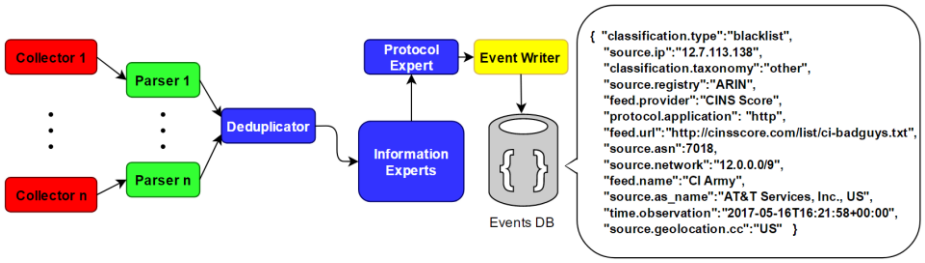


Figura 3. Fluxo de dados do IntelMQ.

4.3 Rules & Blacklists Generator

Este módulo foi desenvolvido para processar os IOAs e os eventos de *blacklist* armazenados em *events DB*, gerando assim as regras e *blacklists* para o SDI, respetivamente. A Figura 4 apresenta este módulo em detalhe. Com base na informação contida no campo *protocol.application* do IOA, o gerador obtém os protocolos e os portos para a regra a criar. De seguida, o gerador cria a regra com a restante informação contida nos IOAs e armazena-a na base de dados de regras (*Rules DB*). A partir dos

eventos de *blacklist*, o gerador extrai o IP e o insere na *blacklist*. O resultado do processamento destes eventos é uma lista negra (*blacklist*) de IPs marcados como maliciosos, a qual é armazenada na base de dados *blacklist* (*Blacklist DB*). Esta lista posteriormente servirá para alimentar o preprocessador de reputação do SDI Snort para verificar se é realizada comunicação para/com IPs maliciosos, a qual pode resultar na contração de vírus e *malware* por parte dos clientes.

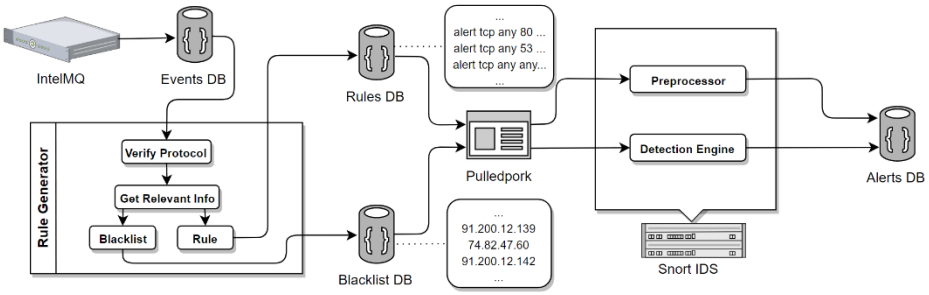


Figura 4. Criação de regras e *blacklists* a partir dos IOAs e eventos de *blacklist*.

4.4 PulledPork e SDI Snort

O PulledPork é um gestor de regras e *blacklists* provenientes de diversas entidades, tanto gratuitas como proprietárias. Como gestor, permite a coexistência de regras de diferentes provedores (ex., *Emerging Threats* [19] e *Talos* [18]), anulando possíveis colisões de regras e IPs destes e das que estão em produção no SDI. Também, gere o fornecimento das regras e IPs ao SDI. Neste sentido, o PulledPork recolhe as regras e as *blacklists* geradas pelo *Rules & blacklists Generator*, evita as possíveis colisões que estas podem originar e reencaminha-as para o Snort.

O Snort analisa todo o tráfego que flui entre a Internet e a intranet da instituição, utilizando estas regras e *blacklists*. As regras contêm padrões sobre ameaças que permitem identificar intrusões no tráfego analisado. As *blacklists* são listas de IPs com má reputação e conhecidos por realizarem diversas ações maliciosas (ex., DDoS). O preprocessador do Snort utiliza estas listas para identificar se houve algum tipo de comunicação por parte destes IPs.

De forma conjunta, o Snort utiliza as regras de alerta geradas pelo *Rule Generato*r no motor de alertas e os IP's *blacklist* no preprocessador de reputação.

5 Avaliação Experimental

A avaliação da solução proposta foi efetuada em ambiente real, na Reitoria da Universidade de Lisboa, na análise do tráfego da rede interna, durante 8 dias. Foram analisadas as redes do *Datacenter*, *Serviços Centrais da Reitoria* (utilizadores) e *Eduroam*. O desempenho da solução foi medido pela contabilização dos alertas advindos das regras e *blacklists* gerados pela solução e confirmação de parte destes por inferência de características conhecidas.

5.1 Set-up e performance da solução

A máquina utilizada na avaliação experimental foi um HP ProLiant DL360 G6 com 2 processadores Intel Xeon CPU X5550 a 2.67GHz, 12Gb de RAM, 165GB de disco rígido e 3 interfaces de rede, sendo uma das interfaces para ligação de fibra ótica a 10Gb/s e as restantes duas de *ethernet* a 1Gb/s. Foi instalado na máquina o sistema operativo *Security Onion*, uma distribuição Linux munida de diversas ferramentas de segurança e análise, entre elas o Snort. O IntelMQ foi instalado numa máquina virtual com 4 Gb de RAM e 17 GB de armazenamento.

De facto, só quando uma solução é colocada em produção é que se consegue validar, seja em configuração, qualidade ou performance, e apurar o quanto é necessário afinar [8]. O set-up da solução não fugiu a esta regra, verificando-se que o *hardware* da solução era insuficiente para aguentar as 3 redes. Perante este constrangimento foi necessário a manutenção constante para possibilitar a recolha dos dados da melhor forma. Contudo, a percentagem de pacotes perdidos não ultrapassou em média os 0.1%, sendo que a pior perda registada foi de 5%, em hora de ponta e tendo o sistema aproximadamente 12.000 regras e 16.500 IPs em funcionamento.

5.2 Registo de incidentes

Durante o período de avaliação da solução foram usados dois conjuntos distintos de regras e *blacklists*. Nos primeiros 3 dias, onde decorreu o teste da solução, foram utilizados os mesmos conjuntos de regras e *blacklists*. Nos restantes 5 dias foram sendo geradas novas regras e *blacklists* todos os dias, e atualizado o Snort. Durante o tempo em que a solução recolheu resultados, foram geradas em média 5.290 regras por dia e 14.590 entradas de *blacklist*.

Foram registados aproximadamente 5.5 milhões de incidentes. A Tabela 2 apresenta este valor estratificado pelas diversas categorias de incidentes. O tráfego gerado por IPs pertencentes a *blacklists* totaliza aproximadamente 70% dos incidentes. Este resultado pode advir do facto do preprocessador não permitir mais nenhum tipo de processamento após haver uma correspondência na *blacklist*.

Os eventos registados pelo processador das regras correspondem aos restantes 30% do total dos incidentes. Destes, a comunicação SSH por um IP considerado malicioso foi a categoria que registou mais alertas. Na categoria da comunicação com domínios maliciosos foi detetada padrões que possivelmente eram referentes a *bots* ou C&C, contudo, dada a larga quantidade de dados não foi humanamente possível confirmar se eram ou não de facto. O mesmo sucedeu com os alertas de IMAP, SMTP e FTP. Alguns dos pedidos SIP OPTIONS foram verificados e provinham maioritariamente de uma *botnet* denominada *Friendly-scanner* [24] que utiliza como recurso a ferramenta *SIPVicious* [25]. O mesmo foi verificado nos pedidos de SIP REGISTER. Os restantes alertas foram gerados por comunicações de IPs maliciosos. Alguns pedidos SNMP foram também confirmados, onde foi possível verificar pedidos realizados com os comandos *get* e *get-next*, cujo objetivo é recolher informação detalhada de equipamentos que poderá ser utilizada em ataques a estes mesmos equipamentos.

Evento	Nº Alertas
Tráfego gerado por um IP em Blacklist	3.779.638
Comunicação SSH por parte de um IP malicioso	1.575.251
Comunicação com Domínio malicioso	45.081
Pedido SIP OPTIONS por IP malicioso	22.126
Comunicação de IP malicioso a servidores WEB	11.770
Comunicação IMAP por parte de um IP malicioso	7.230
Comunicação SMTP por parte de um IP malicioso	4.775
Comunicação FTP por parte de um IP malicioso	1.536
Pedido SIP REGISTER por IP malicioso	1.328
Comunicação Telnet por parte de um IP malicioso	1.003
Comunicação SNMP por parte de um IP malicioso	791
Comunicação SIP por parte de um IP malicioso	369
Total	**5.450.898**

Tabela 2. Registo de incidentes por categoria de evento.

A Tabela 3 mostra o número total de alertas gerados por cada um dos 8 dias. Em média, foram gerados aproximadamente 540.000 alertas nos sete primeiros dias. No último dia houve um aumento significativo de alertas, que possivelmente é justificado com o dia em que surgiu o surto do *ransomware Petya* [28].

Dia 1	Dia 2	Dia 3	Dia 4	Dia 5	Dia 6	Dia 7	Dia 8	Total
410.795	523.005	411.344	505.541	472.309	840.062	605.033	1682.809	**5.450.898**

Tabela 3. Número de alertas por dia.

Relativamente à informação gerada pelas regras de *blacklist*, foram registadas os portos de origem e de destino dos pacotes *blacklist* para melhor compreensão do objetivo da comunicação. A Tabela 4 apresenta estes dados.

Porto de origem	Nº Alertas
62572	126.073
0	125.324
10000	75.689
55763	56.497
52638	44.592
53671	42.513
65535	35.921
58022	35.411
52925	30.241
6000	23.158
Total	**59.5419**

Porto de destino	Nº Alertas
23	568.242
1433	469.893
22	386.229
43526	125.788
0	125.315
5060	108.238
1900	95.962
80	87.001
1443	81.796
443	70.105
Total	**2.118.569**

Tabela 4. Número de eventos *blacklist* registados por porto origem e destino.

Os portos de origem (parte esquerda da tabela) são maioritariamente dinâmicos com a exceção do 6000 e do 0. O porto 6000 é respeitante ao sistema X11 e alguns tipos de *trojans* [26], enquanto o porto 0 é utilizado para recolha de informação, apesar de ser considerado proibido [27]. Analisando os portos de destino (parte direita da tabela) conclui-se que o serviço mais acedido pelos IPs em *blacklist* foi o Telnet (23),

seguidamente dos portos utilizados pelos serviços MySQL (1433) e SSH (22). Curiosamente o quarto porto mais acedido (43526) é um porto utilizado dinamicamente, podendo revelar que uma máquina foi comprometida e está a comunicar com um IP de *blacklist*. Os restantes portos mais acedidos foram os de comunicação VoIP (5060), SSDP (1900), HTTP (80), IES (1443) e HTTPS (443). Estes resultados mostram que os IPs em *blacklist* devem estar bloqueados em qualquer instituição devido à quantidade de acessos indevidos a serviços, como os de SSH e Telnet.

6 Trabalho Relacionado

Zeng *et al.* apresentaram uma arquitetura híbrida (baseada em rede e em host) para a deteção de *botnets* que tinha como principal caraterística a interseção de dados provenientes da rede e dos computadores. Numa primeira fase, a arquitetura, através da análise de tráfego, identifica computadores suspeitos, cujo comportamento seja muito diferente dos restantes. Numa segunda fase, a arquitetura através de análise baseada em host valida se esses computadores são de facto suspeitos. Este método permite uma avaliação mais precisa, pois para além de analisar o comum comportamento de coordenação intrínseco às *botnets*, também realiza uma análise ao seu comportamento como um só [22].

Sperotto *et al*. propuseram um SDI, que por questões de performance e em redes de alta velocidade (ex., 10Gbps), analisa o fluxo do tráfego na rede em vez de analisar o pacote todo. Contudo, o SDI só consegue detetar ataques de negação de serviço, *scans*, *worms* e *botnets*, devido ao facto de somente analisar os cabeçalhos dos pacotes. Contrariamente à solução proposta neste artigo, este tipo de SDI não consegue detetar ataques de *Phishing*, por exemplo, uma vez que é necessário analisar o conteúdo do pacote [23].

Um survey relativo à deteção de *botnets* mostra que, apesar do avanço em alguns métodos de deteção, poucos foram bem-sucedidos e que a técnica mais usada na deteção é a baseada em anomalias. Contudo, continua a existir uma grande dificuldade no desmantelamento de *botnets* porque os *botmasters* continuam a evoluir as técnicas de mascaramento [10]. Aviv *et al.* propõe uma cooperação de diversas entidades para desenvolvimento de mecanismos para deteção de *botnets*, uma vez que estas entidades possuem informação confidencial [8].

Para além do funcionamento *standard* dos SDI, existem SDI específicos para um determinado tipo de ataque e que utilizam técnicas que não as *standard*. O FeatureSmith é um sistema de deteção de *malware* em aplicações android cujas funcionalidades assentam em mineração de dados. Para a aplicação de mineração de dados o sistema coletou dados sobre *malware* em android de 1068 artigos científicos do *google scholar*, onde através do relacionamento semântico com o comportamento de *malware* e do seu mapeamento para funcionalidades o sistema conseguiu uma taxa de deteção de 92.5% de verdadeiros positivos, com uma taxa de 1% de falsos positivos [21]. Ao contrário do FeatureSmith, a solução apresentada baseia-se em deteção de padrões intrusões conhecidos, recolhidos de eventos OSINT e processados para criação de regras e identificação de endereços IP maliciosos, que serão usados pelo SDI.

O MISP é uma plataforma colaborativa de partilha de informação de ameaças de segurança. A sua arquitetura baseia-se em organização e comunidade. Estes dois grupos partilham informação de forma simples e controlada com o objetivo de alertar as entidades pertencentes à organização ou a determinada comunidade [20]. Tal como a solução apresentada, esta plataforma consegue gerar regras para SDI, contudo, quando aplicados num sistema em produção, a sua performance torna a sua utilização impraticável, não tendo portanto a utilidade pretendida.

7 Conclusão

Este artigo apresentou uma arquitetura de melhoramento para deteção de incidentes por um sistema detetor de intrusões (SDI), baseado em conhecimento extraído de eventos de segurança OSINT. A arquitetura é composta por três componentes, que combinados, permitem a extração de conhecimento OSINT para geração de novo conhecimento sob a forma de regras de SDI e *blacklists*, e a deteção de incidentes usando este conhecimento.

A arquitetura foi implementada e avaliada em ambiente real, na Reitoria da Universidade de Lisboa, mostrando-se efetiva na deteção e registo de incidentes.

8 Agradecimentos

Este trabalho foi parcialmente suportado pela EC através do projeto H2020-700692 (DiSIEM) e pelos fundos nacionais através da Fundação para a Ciência e a Tecnologia (FCT) com referência para UID/CEC/00408/2013 (LaSIGE).

9 Bibliografia

1. P. Bächer, T. Holz, M. Kötter, G. Wicherski, 30/11/2016.: “Know your Enemy: Tracking Botnets,” Honeynet, https://www.honeynet.org/book/export/html/50.
2. I. Shakeel, 1/12/2016.: “Evolution in the World of Cyber Crime,”, http://resources.infosecinstitute.com/evolution-in-the-world-of-cyber-crime/#gref.
3. R. Puri, Bots & Botnets: An Overview, SANS Institute, 2003.
4. SearchSecurity, 12/1/2017.: "Details emerging on Dyn DNS DDoS attack, Mirai IoT botnet", http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet.
5. M. Rouse, 5/12/2016.: “Metamorphic and polymorphic malware,”, http://searchsecurity.techtarget.com/definition/metamorphic-and-polymorphic-malware.
6. J. M. Butle, Finding Hidden Threats by Decrypting SSL, SANS Institute, 2013.
7. G. Bruneau, The History and Evolution of Intrusion Detection, SANS Institute, 2001.
8. A. H Haeberlen, A. Aviv: Challenges in Experimenting with Botnet Detection Systems. In: Proceedings of the 4th Conference on Cyber Security Experimentation and Test. CSET’11. pp 6-15 (2011).

Indicadores de Segurança em Plataformas de Monitorização

Raimundo Chipongue, Hugo Miranda, and António Broega

Faculdade de Ciências da Universidade de Lisboa, Campo Grande, 1749-016 Lisboa
fc48807@alunos.ciencias.ulisboa.pt,{hamiranda,ajbroega}@ciencias.ulisboa.pt

Resumo A separação entre administradores de sistemas e redes (TI) e especialistas de segurança afeta a capacidade e tempo de resposta a incidentes de muitas organizações. Esta separação é também observável na limitada oferta de módulos relacionados com segurança nos repositórios públicos de plataformas de monitorização. No entanto, a integração da visão de administração de TI com a da equipa de segurança, por exemplo nas consolas das plataformas de monitorização, pode contribuir para um diagnóstico mais eficaz e para a deteção precoce e resolução de incidentes. Este artigo discute a contribuição das ferramentas de monitorização no colmatar desta separação e propõe um conjunto de módulos para a plataforma Nagios, que procuram proativamente vulnerabilidades conhecidas e comportamentos anómalos, indicadores de potenciais problemas de segurança. A avaliação dos módulos em ambiente de produção e a sua aprovação pela comunidade de editores do repositório oficial de módulos do Nagios fazem-nos concluir que é possível usar a flexibilidade destas ferramentas para monitorização conjunta de sistemas, redes e eventos de segurança, facilitando e acelerando a sua deteção e resolução.

Palavras-chave: Monitorização, Segurança, Nagios.

1 Introdução

A segurança é apenas mais um dos elementos que pode afetar a disponibilidade e desempenho de toda uma infraestrutura de sistemas de informação de uma organização. Em redes corporativas com um número considerável de equipamentos, a tarefa dos administradores de tecnologias de informação (TI) deixa de ser trivial. Os sistemas de monitorização são uma ferramenta importante na deteção de comportamentos fora dos padrões previamente estabelecidos, gerando notificações em tempo real, contribuindo desta forma para acelerar a deteção e resolução de problemas.

Culturalmente, existe uma separação entre administradores de TI e técnicos de segurança ou, nalguns casos, a um isolamento das funções dos segundos apenas numa das áreas cobertas pelos primeiros. Esta separação, apesar de compreensível do ponto de vista organizacional, não é de todo benéfica pois pode levar a um atraso no diagnóstico ou mesmo a que um incidente tenha duas interpretações contraditórias, possivelmente levando à tomada de ações contraproducentes. Por

exemplo, apesar de se tratar de uma ação maliciosa, a ocorrência de um ataque de negação de serviço pode ser interpretado inicialmente como uma limitação de desempenho ou perda de disponibilidade pelo administrador do sistema.

1.1 Motivação

As plataformas de monitorização de TI desempenham tipicamente um duplo papel. Por um lado, atuam como agente profilático, permitindo detetar antecipadamente problemas que se não forem corrigidos atempadamente poderão resultar em incidentes sérios. Por outro, facilitam as operações de diagnóstico de incidentes, apresentando conjuntos de sintomas que se corretamente interpretados pelos administradores de TI aceleram a sua resolução.

A maioria das plataformas de monitorização não favorece a integração das perspetivas dos administradores de TI e dos especialistas de segurança informática. Em repositórios de algumas das mais populares plataformas de monitorização, como por exemplo, o Nagios Exchange,[1] os indicadores de segurança são abordados de forma relativamente marginal. Este estudo contribui para aproximar as duas perspetivas, ao encorajar a que os administradores de TI e especialistas de segurança tenham uma visão única do sistema através da plataforma de monitorização Nagios. O objetivo é por um lado, alargar o carácter profilático das ferramentas de monitorização aos sintomas evidenciados por falhas de segurança e por outro facilitar uma análise cruzada da informação de diagnóstico, diminuindo o impacto dos incidentes.

Este trabalho é motivado por casos como o recente ciberataque à escala global com Ransomware *WannaCry* [1], ocorrido em Maio de 2017. Apesar de explorar uma vulnerabilidade entretanto corrigida nas atualizações do sistema operativo Windows o ataque teve um impacto considerável que poderia ter sido mitigado em sistemas com plataformas de monitorização a verificarem as atualizações aplicadas aos computadores pessoais.

O artigo apresenta uma contribuição para a comunidade de utilizadores da plataforma de monitorização Nagios, descrevendo um conjunto de módulos que permitem sinalizar vulnerabilidades de segurança nos sistemas de uma organização. Estes módulos foram recentemente submetidos e aprovados pelos editores do Nagios Exchange encontrando-se publicamente disponíveis nesse repositório.

2 Trabalho Relacionado

Existe uma série de projetos disponíveis que abordam as contribuições das plataformas de monitorização nos processos de administração de uma infraestrutura de TI. Muitos destes projetos cingem-se ao estudo e utilização das plataformas como ferramenta para auxiliar a administração [22,18,2] ou apontando os pontos fortes e fracos [19]. O trabalho apresentado em [12] distingue-se dos restantes por se focar numa comparação de diferentes aproximações.

[1] `https://exchange.nagios.org`

É cada vez mais comum para grandes empresas o uso de ferramentas de Gestão de Eventos e Informação de Segurança (*Security Information and Event Management*, SIEM) para auxiliar a monitorização segura de infra-estruturas de redes e sistemas. Os SIEM são importantes componentes em TI, capazes de recolher, processar, analisar, armazenar e correlacionar registos (logs) ou fontes de informação de eventos de segurança, produzidos por ferramentas como Sistemas de Deteção de Intrusões (*Intrusion Detection Systems*, IDS), Anteparas de Segurança (*Firewalls*), Gestores de Identidades de Acesso (*Identity Access Manager*, IAM) [9], entre outros. São usados para ajudar a responder rapidamente a ataques e organizar dados de log. Entretanto a implementação e manutenção de um SIEM é um processo bastante complexo e com elevados custos, o que torna inviável para maioria das pequenas e médias empresas.

Por outro lado, apesar de alguns sistemas de monitorização como o Nagios, poderem ser utilizados como fontes de informação dos SIEM, a sua correta configuração, aliada ao desenvolvimento de módulos específicos transforma estas plataformas em alternativas viáveis e de baixo custo aos SIEM's.

Apesar da variedade de trabalhos publicados, não é fácil encontrar trabalhos que partilhem o foco deste artigo. Ou seja, que abordem as plataformas de monitorização, e que tirem proveito das suas potencialidades para gerar indicadores de segurança e consequentemente incrementar a segurança dos sistemas.

2.1 Nagios

O Nagios[2] [6] é uma popular ferramenta de monitorização, distribuída ao abrigo da licença GPL. Usa um núcleo que recorre a plugins para executar tarefas específicas, exibindo os resultados numa interface Web. Além disso, dispõe de um serviço de notificações modular, ao qual podem ser associadas ferramentas de envio de correio eletrónico, SMS, ou qualquer outro meio definido. O acompanhamento exaustivo, administração centralizada, notificações em tempo real, correção automática de problemas (nalguns casos), disponibilização de relatórios, estatísticas e fácil integração, são alguns de seus benefícios.

Plugins Os módulos ou plugins são pequenos programas externos [5] responsáveis pela realização das operações de diagnóstico e pela interpretação dos resultados. A principal informação entregue por um módulo ao núcleo do Nagios é um de quatro possíveis estados da entidade monitorizada [20]:

OK quando nenhuma anomalia for detetada;
WARNING estado intermédio espectávelmente apresentado quando o indicador ainda não atingiu o limite normal de funcionamento;
CRITICAL quando o indicador exceder o limite máximo de funcionamento ou não é recebida resposta;
UNKNOWN quando ocorre um erro na execução do módulo, incapacidade de obter informação por razões conhecidas ou incorreção na definição de parâmetros.

[2] `https://www.nagios.org/`

Agentes Para suportar a execução remota de operações de monitorização, o Nagios recorre a agentes, instalados no sistema remoto e que em resposta a uma invocação executam o plugin designado pela plataforma. A comunicação entre a plataforma pode ser realizada utilizando qualquer protocolo, sendo os principais o standard *Simple Network Management Protocol* (SNMP) [7,18], e o *Nagios Remote Plugin Executor* (NRPE), cocretizado pelo agente com o mesmo nome para o sistema operativo Linux e pelos agentes *Nagios Service Check Acceptor* (NSCA) e *NSClient++* para SOs Windows [5,18,21].

3 Módulos Desenvolvidos

Os módulos desenvolvidos têm a capacidade de monitorizar sistemas e serviços e produzir alertas de segurança através da pesquisa e análise de estados que sinalizem a subversão das políticas de segurança dos sistemas.

A seleção de módulos teve por base um levantamento de requisitos junto da equipa de administração de redes/sistemas e segurança da infraestrutura de TI duma organização de ensino com mais de 5000 utilizadores e com uma grande variedade de requisitos, incluindo por exemplo a criação de web sites específicos, a criação de redes próprias e o alojamento de equipamentos de centro de dados. Esta realidade permitiu que o processo de idealização, desenvolvimento e testes tivesse o acompanhamento da equipa de administração, dando feedback sobre as opções tomadas, sugerindo alterações, realizando testes e validando as opções tomadas. No principio deste processo foram identificados os procedimentos mais comuns e as vulnerabilidades cuja verificação estava já sistematizada mas era despoletada e realizada com procedimentos que obrigavam à intervenção da equipa de administração.

Os módulos foram desenvolvidos usando a linguagem de programação Python, e recorrendo a utilitários Linux e ferramentas de código aberto.

Monitorização do Ficheiro de Log do Servidor Apache O Apache[3] é um servidor Web bastante popular, usado principalmente em sistemas operativos Linux. Ao responder às solicitações, este servidor regista num ficheiro de *log* o código de resposta HTTP devolvido ao cliente [14]. Os códigos HTTP estão agrupados em 5 classes conhecidas como 1xx, 2xx, 3xx, 4xx e 5xx.

Os códigos da classe 4xx sinalizam erros nos pedidos por parte do cliente. Em particular os códigos 401, 403 e 404 sinalizam respetivamente a tentativa de acesso a conteúdo não autorizado, conteúdo para o qual não foram apresentadas credenciais de autenticação válidas ou tentativa de acesso a conteúdo inexistente. Registos repetidos de acessos que gerem estes códigos sugerem uma provável tentativa de subversão das políticas de segurança por parte de um ou um conjunto de clientes que se encontrem a averiguar a existência de vulnerabilidades na configuração do servidor.

[3] `http://www.apache.org/`

O módulo Nagios ***check_apache_status*** foi desenvolvido para auditar o ficheiro de log do servidor Apache, procurando ocorrências dos códigos 401, 403 e 404. Em caso de deteção são gerados alertas com os estados *WARNING* ou *CRITICAL*, dependendo da parametrização definida que contabiliza a quantidade máxima aceitável de códigos no ficheiro de *log* e por endereço IP. O módulo possibilita ainda a definição da quantidade de registos a auditar.

Monitorização dos Programas Instalados O controlo do conjunto de ficheiros executáveis instalados nas plataformas Linux é uma tarefa complexa devido à sua grande quantidade e distribuição por um conjunto de diretórios. Adicionalmente, a quantidade de programas pode variar a cada instalação ou atualização de pacotes de software ou do sistema operativo. Esta arquitetura torna os sistemas Linux vulneráveis à instalação de troianos ou *backdoors* que podem passar facilmente despercebidos a um administrador de TI.

O módulo ***check_app*** auxilia a equipa de administração por monitorizar a lista de executáveis instalados num conjunto de diretorias predefinidas e configuráveis pelo utilizador. O levantamento é realizado a cada execução e comparado com a listagem obtida na execução anterior, sinalizando anomalias com o estado *CRITICAL*.

Monitorização de Ataques TCP SYN Flood A inundação por pedidos de estabelecimento de ligação (*SYN Flood Attack*) é um tipo de ataque da categoria *Denial of Service* (DoS) ou *Distributed Denial of Service* (DDoS) que consiste na sobrecarga direta da camada de transporte do alvo e indireta na camada de aplicação do modelo OSI. É concretizado pelo envio de sequências de pedidos de estabelecimento de ligação do protocolo TCP (SYN) [10,15].

A deteção destes eventos é feita usando o módulo ***check_synflood***, que monitoriza as ligações de rede em busca de um número de ligações com o estado SYN_RECV acima dos limites definidos pelo utilizador. Consoante os limiares definidos, o módulo pode retornar o estado *WARNING* ou *CRITICAL*.

Monitorização do Conteúdo de Paginas Web A alteração maliciosa de conteúdo disponibilizados através da World Wide Web (WWW) é um ataque bastante usado por *hackers* e vulgarmente designado por *defacement* [13]. O *defacement* tem consequências sobretudo para a imagem da organização, ao expôr a sua vulnerabilidade ou anunciando informação contraditória com os seus objetivos. Tecnicamente, este ataque consubstancia-se na subversão de duas das principais propriedades de segurança da informação, nomeadamente a integridade e a autenticidade.

O módulo ***check_defacement*** monitoriza o conteúdo de páginas web recebidas como argumento. Este módulo verifica se o conteúdo da página apresenta palavras tipicamente utilizadas pelos agentes maliciosos nos ataques de *defacement* sinalizando a sua ocorrência com o estado *CRITICAL*. Por omissão foi definido um conjunto destas palavras, que pode ser alterado através da edição do código ou pela utilização de argumentos na linha de comando.

Monitorização da Correspondência Entre Nomes de Domínio e Endereço IP Associado Um nome de domínio é uma referência de fácil memorização associada a um endereço IP e que serve para identificar um computador na Internet. As informações de correspondência entre nomes de domínios e endereços IP são armazenadas em base de dados de servidores *Domain Name System* (DNS), que asseguram a indicação do endereço certo para a entrega dos pedidos. No entanto, este sistema não é imune a ataques tais como ***DNS cache poisoning***, caraterizado por comprometer a integridade do servidor de DNS. Este ataque é concretizado através da injeção de informações falsas, deturpando a precisão das consultas DNS. Na prática, estes ataques permitem o redirecionamento do tráfego, permitindo a personificação de um endereço legítimo por terceiros [23].

O módulo ***check_dns*** compara a associação entre nomes de domínio e endereços IP em servidores de DNS externos, verificando se a resposta está de acordo com o esperado. O servidor a testar é passado como argumento utilizando-se por omissão o servidor de DNS da Google por ser um dos servidores mais populares. A deteção de incoerências é sinalizada com o estado *CRITICAL*.

Monitorização de Listas Negras As listas negras [16] são uma das mais populares ferramentas de combate ao crescente problema de *spam* e tentativas de *phishing* utilizando o correio eletrónico. As listas são normalmente implementadas sob a forma de registos DNS em domínios geridos por organizações responsáveis pela gestão das listas. Neste modelo, um servidor de correio eletrónico é inserido numa lista negra adicionando o seu endereço IP de forma invertida, por exemplo o endereço 192.168.1.1 é inserido e armazenado como 1.1.168.192.

Para verificar se um determinado endereço IP é confiável, as ferramentas *antispam* efetuam consultas ao DNS, pesquisando o endereço IP do servidor no formato invertido. O conteúdo do registo retornado é irrelevante uma vez que é a sua existência que sinaliza a presença do servidor na lista negra. Estas listas são geridas por entidades privadas, e as políticas de inserção de endereços são suscetíveis a falhas e interpretações erróneas do comportamento por vezes legítimo dos servidores de correio eletrónico.

Tipicamente, mensagens enviadas por servidores em listas negras são liminarmente rejeitadas ou marcadas pelas ferramentas de deteção de SPAM. A presença de um servidor de correio eletrónico legítimo numa lista negra tem por isso consequências negativas no funcionamento da instituição, importa por isso tomar conhecimento e resolver adições incorretas com a maior brevidade possível.

O módulo ***check_dnsbl*** facilita a deteção destes incidentes, ao reproduzir o comportamento de múltiplos filtros *antispam*, verificando nas listas negras a presença do endereço IP recebido como argumento. Por omissão, o módulo pesquisa em 27 das listas negras mais populares. A presença do servidor em pelo menos uma destas listas é assinalada com o estado *CRITICAL*.

Monitorização das Configurações DNSSEC A extensão de segurança de nomes de domínios (*DNSSEC*) usa criptografia assimétrica para assegurar a autenticidade e integridade dos dados manipulados pelo protocolo DNS. O uso do DNSSEC melhora a fiabilidade do sistema, previne ataques *(man-in-the middle, MITM)*, corrige fragilidades do protocolo DNS e diminui a probabilidade de manipulação ilícita da informação [3]. No entanto, quando mal configurado ou com assinaturas expiradas, o DNSSEC dá uma falsa sensação de proteção que pode resultar na exposição não intencional do tráfego.

O módulo ***check_dnssec*** monitoriza o servidor DNS em busca de vulnerabilidades nas configurações do DNSSEC e assinala com o estado *CRITICAL* a sua deteção. Quer o domínio a verificar quer o servidor de DNS utilizado são passados como argumento. Por omissão a validade da informação é realizada no servidor DNS da google (*8.8.8.8*) por ser um dos servidores mais populares.

Monitorização de Ficheiros e/ou Diretorias Num sistema em operação existe um conjunto de ficheiros, por exemplo de configuração, que se espera que se mantenham inalterados por longos períodos de tempo. A verificação da sua alteração pode sinalizar problemas de segurança. No entanto, este processo é particularmente moroso e sujeito a erros pelo que é altamente recomendável optar pela sua monitorização automática.

O módulo ***check_filechange*** alerta para a ocorrência de eventos de alteração do estado de ficheiros e diretorias. O módulo utiliza a ferramenta de monitorização de estados *Inotify-tools* [17], delegando desta forma no próprio sistema operativo o processo de monitorização, contribuindo para a diminuição do consumo de recursos associado a esta tarefa. O módulo recebe como argumentos os ficheiros/diretorias a serem monitorizados, os eventos a sinalizar e o caminho do ficheiro de log onde o *Inotify* registará as deteções. A ocorrência de um ou mais eventos é notificada com o estado *CRITICAL*.

Monitorização de Vulnerabilidades Web Uma vulnerabilidade é um defeito que pode ser explorado por um atacante com o objetivo de subverter a política de segurança [8]. No caso particular de servidores da World Wide Web (WWW) as vulnerabilidades mais comuns devem-se à falta de tratamento pelo serviço de partes do conteúdo do pedido, *cookies* mal configurados, identificadores de sessão expostos no navegador, parâmetros passados no *URL* ou em parâmetros de formulários, por exemplo de *login*.

Um analisador de vulnerabilidades web é uma ferramenta que percorre todo um site WWW tentando identificar pontos onde existam vulnerabilidades bem conhecidas. O módulo ***check_nikto*** usa o *analisador de vulnerabilidades web* NIKTO[4] para realizar estas auditorias. O NIKTO produz relatórios no formato HTML, alertando para a existência de boa parte das vulnerabilidades web mais populares [11]. O site monitorizado é passado como argumento, e opcionalmente podem ser especificados o porto de comunicação, a diretoria onde será armazenado o relatório, o nome e tempo de validade do relatório, assim como os tipos

[4] `https://cirt.net/nikto2`

de vulnerabilidades a serem pesquisadas. O módulo analisa o relatório produzido pelo NIKTO em busca de indicações de vulnerabilidades encontradas e sinaliza a sua deteção com o estado *CRITICAL*.

Monitorização de Portos de Comunicação Porto de comunicação é o número que identifica uma aplicação à qual se destinam os dados de uma comunicação. Desta maneira, os dados são enviados diretamente pelo sistema operativo para a aplicação correspondente. Uma vez que estes são os pontos de entrada e saída de informação na rede, portos abertos fora do controlo dos administradores podem representar uma janela para que utilizadores maliciosos possam aceder ao sistema.

O módulo ***check_open_port*** monitoriza os portos de comunicação e alerta para a abertura de portos que não façam parte da lista de portos autorizados passada como argumento. A identificação destes casos é sinalizada com o estado *CRITICAL*.

Monitorização das Configurações SSL/TLS O *Secure Sockets Layer* (SSL) e o seu sucessor *Transport Layer Security* (TLS) são protocolos criptográficos projetados para garantir autenticidade, confidencialidade e integridade na troca de dados em aplicações cliente/servidor. Estes protocolos usam certificados digitais para cifrar a informação trocada entre aplicações os quais exigem cuidados na sua administração e utilização. Por exemplo, os certificados expiram, podem ser revogados, usar protocolos criptográficos inseguros ou estar mal configurados. O comprometimento de certificados é uma situação de risco elevado por oferecer uma falsa sensação de segurança às partes em comunicação.

O módulo ***check_ssl***, monitoriza as configurações e o estado dos certificados utilizando a API online do *SSLLAB*[5]. Esta API realiza testes às configurações de certificados em busca de vulnerabilidades devolvendo uma classificação numa escala de letras e que indica o nível de gravidade das vulnerabilidades encontradas. A sinalização utiliza os estado *WARNING* ou *CRITICAL* em função da classificação atribuída, dos parâmetros estabelecidos na linha de comando e no prazo de validade do certificado.

Monitorização de Atualizações de CMS Os Content Management Systems (CMS) são ferramentas de produção e gestão de conteúdos web muito populares por oferecem interfaces particularmente amigáveis. A sua popularidade e disseminação tornam-nas também um alvo apetecível de ataques, sendo frequente a identificação de novas vulnerabilidades. O WordPress[6] é um dos CMS mais populares, desenvolvido por uma comunidade aberta, e em constante aperfeiçoamento. Esta comunidade publica frequentemente atualizações que corrigem as vulnerabilidades entretanto identificadas. No entanto, do lado do utilizador a descoberta de novas atualizações nem sempre é um processo trivial, o que leva a

[5] `https://www.ssllabs.com/`
[6] `https://wordpress.org/`

que muitas vezes se utilizem por largos períodos de tempo versões desatualizadas e com vulnerabilidades conhecidas publicamente.

O módulo ***check_wp_update*** auxilia a descoberta de atualizações de segurança deste CMS. Em contraste com outros módulos equivalentes já disponíveis o ***check_wp_update*** utiliza o sistema de ficheiros e não um pedido HTTP para aceder ao ficheiro version.php onde está registada a versão em execução. Apesar de por isso ter que ser executado localmente no servidor, esta versão tem a vantagem de não obrigar à exposição pública da informação da versão em uso e que pode ser utilizada por potenciais atacantes para identificar as vulnerabilidades conhecidas na versão em causa.

Os dados da versão mais recente são obtidos por consulta ao site do WordPress através da API disponibilizada pela equipa de desenvolvimento. A deteção de nova atualização é sinalizada com o estado *CRITICAL*, uma vez que as atualizações mais frequentes deste CMS são as conhecidas por *patchs*, disponibilizadas justamente para correções de *bugs*. Entretanto, apesar do incremento das versões *Major* e *Minor* representar essencialmente a introdução de melhorias nas funcionalidades do software, nada impede que sejam igualmente introduzidas correções de *bugs*.

4 Avaliação

Os testes foram realizados em várias plataformas Linux, tais como *Xubuntu-16-04.01, Lubuntu 16.10, Gentoo 2.2, CentOS 7 e Debian 8*.

No âmbito da avaliação foram simulados diversos cenários que permitiram testar todos os possíveis estados retornados por cada um dos módulos, validando o seu correto funcionamento.

Por exemplo os testes ao módulo *check_syn_flood* foram concretizados usando a ferramenta de ataque DoS/DDoS ***hping3***, através do comando `# hping3 <alvo> -S -p 80 --flood --rand-source`. Enquanto que os testes ao módulo *check_wp_update* passaram por usar versões desatualizadas do CMS WordPress, através do *downgrade* do software, forçando assim a geração de alertas.

Na sequencia dos testes e validação dos módulos por parte dos autores e da equipa de administradores da organização, os módulos passaram a fazer parte da biblioteca utilizada por uma instalação do Nagios a monitorizar 180 servidores (dos quais 160 virtuais) que alojam vários serviços, onde se destacam servidores *World Wide Web* (WWW), correio eletrónico, base de dados, cópias de segurança, *File Transfer Protocol* (FTP), servidores de *Webdav*, *Domain Name System* (DNS), gestão documental e gestão de acessos.

Em ambiente de produção, os módulos encontram-se instalados em maquinas com o sistema operativo Linux CentOS 7, e nos primeiros dias logo após a sua instalação contribuíram para a deteção de alguns problemas:

check_wp_update Este módulo foi o primeiro a corresponder às expetativas, ao alertar para a disponibilização da versão 4.7.4 do WordPress e posteriormente para a versão 4.8, o que permitiu que correções de segurança fossem

instaladas pouco tempo depois de disponibilizadas, reduzindo desta forma a janela de vulnerabilidades para um grande conjunto de web sites alojados no CMS.

check_nikto Este módulo também deixou boas indicações ao detetar vulnerabilidades em alguns dos sítios monitorizados. Destas vulnerabilidades, importa destacar a do tipo *Clickjacking*, que embora não seja considerada vulnerabilidade de grande criticidade, foi prontamente corrigida. Vulnerabilidades deste tipo existem quando é possível um atacante usar camadas transparentes para enganar um utilizador, fazendo com que este clique em um objeto (link ou botão) invisível. Este click pode ser utilizado para a realização de ações maliciosas [4].

check_app, check_synflood, check_filechange e check_open_port Estes módulos estão instalados e a monitorizar parte dos servidores administrados pela organização, não tendo até ao momento detetado qualquer situação anormal.

check_apache_status Este módulo foi igualmente instalado e encontra-se a monitorizar cerca de 32 servidores, sem qualquer deteção até ao momento da escrita deste artigo.

check_defacement Com características reativas encontra-se igualmente instalado e a monitorizar o principal sítio da organização, e embora não tenha até ao momento detetado qualquer situação anormal, foi considerado pela equipa de administração como um dos mais úteis.

check_dns, check_dnssec e check_ssl Estes módulos estão a monitorizar os 7 domínios mais relevantes da organização, não tendo até ao momento detetado qualquer situação anormal.

check_dnsbl Este módulo encontra-se instalado e a monitorizar 3 servidores responsáveis pelo envio de correio eletrónico, tendo por 2 vezes detetado a inclusão de um destes servidores numa lista negra, o que permitiu que se corrigi-se antes que problemas no envio de correios eletrónicos fossem detetados e reportados.

Após submissão ao repositório oficial do Nagios,[7] os módulos foram avaliados, aprovados e disponibilizados publicamente pela equipa de administração deste repositório. Encontram-se por isso disponíveis ao público dentro dos padrões de licenciamento definidos. Os módulos estão também publicamente acessíveis no repositório público github.[8]

5 Conclusões e Trabalho Futuro

Num sistema, as ferramentas de monitorização supervisionam as operações do potencialmente grande número de equipamentos conectados entre si e a Internet. Uma supervisão eficiente e eficaz é fundamental para garantir um serviço

[7] `https://exchange.nagios.org/directory/Plugins/Security/`
[8] `https://github.com/rc48807`

confiável e de alta qualidade. Neste artigo estudamos o Nagios como uma ferramenta de monitorização de código aberto com potencial para melhorar a segurança dos sistemas informáticos através do desenvolvimento de módulos focados na geração de indicadores de segurança.

O trabalho consistiu no desenvolvimento de módulos que verificam o respeito de boas práticas de segurança mas cuja verificação manual é sujeita a erros ou é repetitiva. Estes módulos foram exaustivamente testados, produzindo resultados que indicam ser possível incrementar a segurança das infraestruturas tecnológicas recorrendo à utilização das ferramentas de monitorização.

Os módulos apresentados permitem um acompanhamento automatizado, centralizado e em tempo real de eventos de segurança. Espera-se desta forma contribuir para o aproximar das equipas de administração de TI e de segurança através da partilha de uma plataforma única de monitorização e diagnóstico que ofereça uma visão conjunta e unificada destas duas realidades. Esta combinação contribuiu também para uma colaboração mais eficaz entre estas duas equipas, ambas fundamentais para o correto funcionamento dos serviços. Espera-se um incremento da pro-atividade através da reação atempada a problemas de segurança e uma redução no tempo de resposta a incidentes.

A identificação de requisitos e o desenvolvimento de novos módulos continua. Neste artigo os autores limitam-se em demonstrar e explorar a capacidade das ferramentas de monitorização na melhoria da segurança dos sistemas de acordo com os requisitos identificados por uma organização. Espera-se que num futuro próximo este estudo venha a ter continuação, através do desenvolvimento de novos módulos que dêm continuidade aos princípios que motivaram este trabalho.

Agradecimentos

O trabalho descrito neste artigo foi parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através do financiamento à unidade de investigação LaSIGE (ref. UID/CEC/00408/2013).

Referências

1. CERT-UE Advisory Advisory. Wannacry ransomware campaign exploiting smb vulnerability. Date: May 16 2017.
2. Abhishek Amralkar, Mayank Gaikwad, Rohit Nerkar, Pulkit Gupta, and Mukesh Waghadhare. Monitoring tools. *TechWatch Report*, pages 88–93, 2016.
3. Suranjith Ariyapperuma and Chris J Mitchell. Security vulnerabilities in dns and dnssec. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 335–342. IEEE, 2007.
4. Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A solution for the automated detection of clickjacking attacks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 135–144. ACM, 2010.
5. Wolfgang Barth. *Nagios: System and Network Monitoring*. No Starch Press, San Francisco, CA, USA, 2nd edition, 2008.

Replicação Não Uniforme com Consistência Eventual

Gonçalo Cabrita e Nuno Preguiça

NOVA LINCS & DI, FCT, Universidade NOVA de Lisboa

Resumo A replicação é uma técnica chave no desenho de sistemas distribuídos confiáveis e eficientes. À medida que a informação cresce, torna-se difícil ou até impossível guardar todos os dados em todas as réplicas. Uma solução comum para este problema passa por utilizar técnicas de replicação parcial, onde cada réplica mantém apenas uma parte dos dados. Por consequência, cada réplica consegue apenas responder a um subconjunto das *queries* suportadas. Neste artigo, introduzimos o conceito de replicação não uniforme, onde cada réplica guarda apenas parte dos dados mas consegue responder a todas as *queries* suportadas. Aplicamos este conceito à consistência eventual e aos tipos de dados livres de conflitos. A nossa avaliação mostra que a replicação não uniforme é mais eficiente que a replicação tradicional, conseguindo reduzir o espaço de armazenamento utilizado, a quantidade de dados transmitidos, e aumentando a escalabilidade do sistema.

Palavras-chave: Replicação não uniforme; Replicação parcial; Tipos de dados replicados; Consistência eventual

1 Introdução

Um grande número de aplicações corre sobre infraestruturas de nuvem compostas por múltiplos centros de dados distribuídos geograficamente pelo mundo. Estas aplicações utilizam bases de dados replicadas para obterem latências baixas, alta disponibilidade e tolerância a falhas [6, 7, 9, 11]. Algumas bases de dados fornecem consistência forte [7, 16], dando a ilusão de que apenas uma única réplica existe. Estes sistemas requerem coordenação entre as réplicas para a execução de operações, o que tem impacto na latência e disponibilidade. Outras bases de dados [9, 11] optam por fornecer alta disponibilidade e latência baixa, permitindo a execução local de operações num único centro de dados. Desta forma, conseguem responder aos clientes sem terem de coordenar as operações entre as réplicas, levando a uma latência baixa. Esta aproximação permite a ocorrência de modificações concorrentes, tendo as aplicações de lidar com um modelo de consistência fraca em que é necessário reconciliar réplicas divergentes.

Recentemente, os tipos de dados livres de conflitos [20] (CRDTs) têm sido adotados em várias bases de dados chave-valor distribuídas [5, 21] como um mecanismo de reconciliação seguro. Os CRDTs permitem simplificar a adoção

de modelos de consistência fraca, removendo a necessidade de lidar diretamente com a resolução de conflitos.

Com o aumento da informação mantida nas bases de dados, é impossível ou indesejável manter todos os dados em todas as réplicas. Para além de aplicar técnicas de *sharding* entre múltiplas máquinas em cada centro de dados, é útil manter apenas parte dos dados em cada centro de dados. Em sistemas que adotam um modelo de replicação parcial [8, 19], como cada réplica mantém apenas parte dos dados, só é possível processar um subconjunto de *queries*.

Neste artigo exploramos um modelo de replicação parcial alternativo, o modelo de replicação não uniforme, onde cada réplica mantém apenas parte dos dados mas consegue processar todas as *queries* localmente. A ideia principal é que para alguns objetos replicados, não são necessários todos os dados para responder a uma *query*. Por exemplo, um objeto que mantém o conjunto dos K maiores elementos apenas precisa de manter os K maiores elementos em cada réplica. No entanto, os elementos menores podem ser necessários se uma operação de remoção estiver disponível. Aplicámos este modelo de replicação não uniforme aos CRDTs, formalizando-o para um modelo onde as réplicas sincronizam propagando operações. Adicionalmente, apresentamos um tipo de dados que utiliza este modelo. As nossas avaliações mostram que o modelo de replicação não uniforme leva a uma redução do espaço de armazenamento utilizado e da quantidade de dados transmitidos, e a um aumento da escalabilidade do sistema.

O resto deste artigo está organizado da seguinte forma. A secção 2 discute o trabalho relacionado. A secção 3 descreve o modelo de replicação não uniforme. A secção 4 aplica o modelo a um sistema de consistência eventual. A secção 5 introduz um tipo de dados que segue o modelo. A secção 6 apresenta uma avaliação da solução proposta.

2 Trabalho relacionado

Vários protocolos de replicação foram propostos na última década [2, 10, 14–16, 18, 22]. Em protocolos de replicação total, cada réplica replica a totalidade dos dados. Estes protocolos têm o benefício de que todas as réplicas conseguem responder a todas as *queries* mas requerem que todas as réplicas (ou um quórum) processem todas as operações de escrita. Adicionalmente, cada réplica tem de armazenar todos os dados, o que pode não ser possível ou práticos. A replicação parcial [3, 8, 19] lida com as limitações da replicação total mantendo em cada réplica apenas parte dos dados. Esta aproximação aumenta a escalabilidade do sistema, no entanto como cada réplica mantém apenas parte dos dados só consegue processar um subconjunto de *queries*.

CRDTs [20] são tipos de dados que podem ser replicados, garantindo que todas as réplicas convergem para o mesmo valor após todas as modificações terem sido propagadas para todas as réplicas. A propagação das modificações pode ser feita através do envio das operações executadas (CRDTs baseados em operações) ou através do envio do estado da réplica (CRDTs baseados em estado).

Os CRDTs baseados em deltas [1] são uma evolução dos CRDTs baseados em estado, nos quais durante a sincronização se envia apenas a diferença entre o estado antigo e o mais recente, reduzindo o custo de disseminação das modificações. Os CRDTs computacionais [17] são um extensão dos CRDTs baseados em estado em que o estado de um objeto é o resultado de uma computação sobre as operações executadas – e.g. a média, os K maiores elementos inseridos. Tal como no modelo que propomos neste artigo, as réplicas destes CRDTs não precisam de ter estados equivalentes. O nosso trabalho evolui as ideias inicialmente propostas nos CRDTs computacionais em diversos aspetos, incluindo a definição de um modelo de replicação não uniforme e a sua aplicação a um modelo de consistência eventual onde as réplicas sincronizam por troca de operações.

3 Replicação não uniforme

Consideramos um sistema distribuído assíncrono composto por n nós. Sem perda de generalidade, assumimos que o sistema replica um único objeto. O objeto suporta uma interface composta por um conjunto de operações de leitura, $\mathcal{Q}$, e um conjunto de operações de escrita, $\mathcal{U}$. O estado que resulta da execução de uma operação o num estado s representa-se por $s \bullet o$. Para uma operação de leitura, $q \in \mathcal{Q}$, $s \bullet q = s$. O resultado da aplicação de uma operação $o \in \mathcal{Q} \cup \mathcal{U}$ num estado $s \in \mathcal{S}$ representa-se por $o(s)$.

Definimos o estado do sistema replicado como um tuplo $(s_1, s_2, \ldots, s_n)$, onde s_i refere-se ao estado da réplica i. O estado das réplicas é sincronizado por um protocolo de replicação que envia mensagens entre os nós do sistema e atualiza o estado das réplicas. Não consideramos nenhum protocolo específico, visto que o nosso modelo pode ser aplicado a diferentes protocolos.

Dizemos que o sistema encontra-se num estado quiescente se, para um dado conjunto de operações, o protocolo de replicação tiver propagado todas as mensagens necessárias para sincronizar todas as réplicas, i.e., mensagens posteriormente enviadas pelo protocolo de replicação não modificarão o estado das réplicas. De uma forma geral, os protocolos de replicação tentam garantir uma propriedade de convergência, onde o estado de quaisquer duas réplicas é equivalente num estado quiescente.

Definição 1 (Estado equivalente). *Dois estados s_i e s_j, são equivalentes, $s_i \equiv s_j$, sse o resultado da execução de uma operação $o_n \in \mathcal{Q} \cup \mathcal{U}$ após a execução de uma qualquer sequência de operações $o_1, \ldots, o_{n-1}$ com $o_1, \ldots, o_n \in \mathcal{Q} \cup \mathcal{U}$ em ambos os estados é igual, i.e., $o_n(s_i \bullet o_1 \bullet \ldots \bullet o_{n-1}) = o_n(s_j \bullet o_1 \bullet \ldots \bullet o_{n-1})$*

Esta propriedade é garantida pela maioria dos protocolos de replicação, independentemente de fornecerem um modelo de consistência forte ou fraca [13, 14, 22]. Esta propriedade não requer que o estado interno das réplicas seja igual, mas apenas que as réplicas retornem os mesmos resultados para a execução de uma qualquer sequência de operações. Neste artigo, propomos relaxar esta propriedade requerendo apenas que a execução de operações de leitura retorne o mesmo valor. Denominamos esta propriedade de *observavelmente equivalente* e definimo-la formalmente da seguinte forma.

Definição 2 (Estados observavelmente equivalentes). *Dois estados, s_i e s_j, são observavelmente equivalentes, $s_i \stackrel{o}{\equiv} s_j$, sse o resultado da execução de uma operação $o \in \mathcal{Q}$ em ambos os estados for igual, i.e., $o(s_i) = o(s_j)$.*

De seguida definimos um sistema não uniforme como um sistema que garante apenas que as réplicas convergem para um estado observável equivalente.

Definição 3 (Sistema replicado não uniforme). *Dizemos que um sistema replicado é não uniforme se o protocolo de replicação garante que num estado quiescente, o estado de quaisquer duas réplicas é observavelmente equivalente.*

4 Consistência eventual não uniforme

4.1 Modelo do sistema

Consideramos um sistema distribuído assíncrono composto por n nós, onde os nós podem exibir falhas *fail-stop* mas não falhas bizantinas. Assumimos que é utilizada uma ligação fiável e que está disponível um sistema de comunicação com uma primitiva de comunicação, $mcast(m)$, que pode ser usada por um processo para enviar uma mensagem para todos os outros processos do sistema.

Um objeto é definido como um tuplo $(\mathcal{S}, s^0, \mathcal{Q}, \mathcal{U}_p, \mathcal{U}_e)$, onde $\mathcal{S}$ é o conjunto de estados validos do objeto, $s^0 \in \mathcal{S}$ é o estado inicial do objeto, $\mathcal{Q}$ é o conjunto de operações de leitura, $\mathcal{U}_p$ é o conjunto de operações *prepare-update* e $\mathcal{U}_e$ é o conjunto de operações *effect-update*. Uma operação de leitura executa apenas na réplica onde a operação é invocada, a origem, e não tem efeitos colaterais, i.e., não altera o estado do objeto. Quando uma aplicação pretende atualizar o estado de um objeto, deve invocar uma operação *prepare-update*, $u_p \in \mathcal{U}_p$. Uma operação u_p executa apenas na origem, não tem efeitos colaterais e gera uma operação *effect-update*, $u_e \in \mathcal{U}_e$. Na origem, u_e executa imediatamente após u_p.

Como apenas as operações *effect-update* podem alterar o estado do objeto, para raciocinar sobre a evolução das réplicas restringimos a análise a estas operações. A execução de uma operação u_p gera uma instância de uma operação *effect-update*. Por simplicidade, chamamos a instâncias de operações simplesmente operações. Sendo O_i o conjunto de operações geradas no nó i, o conjunto de operações geradas numa execução, é $O = O_1 \cup \ldots \cup O_n$.

4.2 Consistência eventual não uniforme

Para uma qualquer execução, sendo O o conjunto de operações da execução, um sistema replicado fornece *consistência eventual* sse num estado quiescente: (i) todas as réplicas executaram todas as operações de O; e (ii) os estados de qualquer par de réplicas são equivalentes.

Uma condição suficiente para garantir a primeira propriedade é propagar todas as operações geradas utilizando difusão fiável e executar qualquer operação recebida. Uma condição suficiente para garantir a segunda propriedade passa

por apenas permitir operações comutativas. Assim, se todas as operações comutarem, a execução de uma qualquer serialização de O no estado inicial do objeto resulta num estado equivalente. A partir de agora, assumimos que todas as operações comutam. Como todas as serializações de O são equivalentes, denotamos a execução de uma serialização de O num estado s como $s \bullet O$.

Para uma dada execução, sendo O o conjunto de operações da execução, um sistema replicado fornece *consistência eventual não uniforme* sse num estado quiescente: (i) o estado de qualquer réplica é observavelmente equivalente ao estado obtido pela execução de uma serialização de O; e (ii) os estados de qualquer par de réplicas são observavelmente equivalentes. Para um dado conjunto de operações numa execução O, dizemos que $O_{core} \subseteq O$ é o conjunto de operações essenciais de O sse $s^0 \bullet O \overset{\circ}{\equiv} s^0 \bullet O_{core}$. Definimos o conjunto de operações irrelevantes para o estado final das réplicas da seguinte forma: $O_{masked} \subseteq O$ é o conjunto de operações irrelevantes de O sse $s^0 \bullet O \overset{\circ}{\equiv} s^0 \bullet (O \setminus O_{masked})$.

Teorema 1 (Condições suficientes para NuEC). *Um sistema replicado fornece consistência eventual não uniforme (NuEC) se, para um dado conjunto de operações O, as seguintes condições forem verdadeiras: (i) cada réplica executa um conjunto de operações essenciais de O; e (ii) todas as operações comutam.*

Demonstração. Tendo a definição de operações essenciais de O, e sabendo que todas as operações comutam, podemos dizer que se qualquer réplica executa um conjunto de operações essenciais, então o estado final de todas as réplicas é igual e equivalente ao estado obtido pela execução de uma qualquer serialização de O.

4.3 Protocolo para consistência eventual não uniforme

Agora utilizamos as condições anteriormente definidas para criar um protocolo de replicação que tenta minimizar as operações propagadas para outras réplicas. A ideia principal é evitar propagar operações que façam parte do conjunto de operações irrelevantes. O desafio é fazê-lo utilizando apenas informação local, que inclui apenas um subconjunto das operações executadas.

O algoritmo 1 apresenta o pseudocódigo de um algoritmo utilizado para fornecer *consistência eventual não uniforme*. O algoritmo não garante a durabilidade das operações irrelevantes, sendo esta questão discutida na Secção 4.4.

Para além do estado do objeto, o algoritmo mantém também três conjuntos de operações: (i) $log_{coreLocal}$, o conjunto de operações essenciais geradas localmente na réplica que ainda não foram propagadas para outras réplicas; (ii) log_{local}, o conjunto de operações geradas localmente na réplica que ainda não foram propagadas para outras réplicas; e (iii) log_{recv}, o conjunto de operações que foram propagadas para todas as réplicas, incluindo as operações geradas localmente.

Quando uma operação é gerada, a função *execOp* é invocada. Esta função adiciona a nova operação ao conjunto de operações locais e atualiza o estado do objeto local. Se a nova operação tem um impacto no estado observável do objeto, então esta é também adicionada ao conjunto de operações locais essenciais.

A função *sync* é utilizada para propagar operações locais para réplicas remotas. A função começa por calcular que operações novas devem ser propagadas,

Algoritmo 1 Algoritmo de replicação para consistência eventual não uniforme

```
 1: S : state: initial s^0                                          ▷ Estado do objeto
 2: log_recv : set of operations: initial {}
 3: log_local : set of operations: initial {}                       ▷ Operações locais não propagadas
 4: log_coreLocal : set of operations: initial {}                   ▷ Operações essenciais locais não propagadas
 5:
 6: execOp(op): void                                                ▷ Nova operação gerada localmente
 7:   if hasImpact(op, S) then
 8:     log_coreLocal = log_coreLocal ∪ {op}
 9:   log_local = log_local ∪ {op}
10:   S = S • op
11:
12: opsToPropagate(): set of operations                             ▷ Operações a serem propagadas
13:   ops = maskedForever(log_local ∪ log_coreLocal, S, log_recv)
14:   log_local = log_local \ ops
15:   log_coreLocal = log_coreLocal \ ops
16:   opsImpact = log_coreLocal ∪ hasObservableImpact(log_local, S, log_recv)
17:   opsPotImpact = mayHaveObservableImpact(log_local, S, log_recv)
18:   return opsImpact ∪ opsPotImpact
19:
20: sync(): void                                                    ▷ Propaga operações locais para réplicas remotas
21:   ops = opsToPropagate()
22:   mcast(ops)
23:   log_coreLocal = {}
24:   log_local = log_local \ ops
25:   log_recv = log_recv ∪ ops
26:
27: on receive(ops): void                                           ▷ Processa operações remotas
28:   S = S • ops
29:   log_recv = log_recv ∪ ops
```

realiza a difusão dessas operações, e finalmente atualiza o conjunto local de operações. Quando uma réplica recebe um conjunto de operações (linha 27), esta atualiza o estado local e o conjunto de operações propagadas para todas as réplicas.

A função *opsToPropagate* aborda o desafio de decidir que operações necessitam de ser propagadas para outras réplicas. Para este fim, dividimos as operações em quatro grupos: (i) o conjunto de operações que serão sempre irrelevantes; (ii) o conjunto de operações essenciais ($opsImpact$); (iii) o conjunto de operações que podem ter impacto no estado observável dependendo das operações de outras réplicas que não foram propagadas ($opsPotImpact$); e (iv) as restantes operações que podem ter impacto no estado observável do objeto dependendo das operações que poderão ser executadas e que sejam propagadas para todas as réplicas.

Para demonstrar que o algoritmo pode ser usado para fornecer consistência eventual não uniforme, temos de provar a seguinte propriedade.

Teorema 2. *O algoritmo 1 garante que num estado quiescente todas as réplicas receberam todas as operações essenciais.*

Demonstração. Para demonstrar esta propriedade, temos de provar que não existe nenhuma operação não propagada que seja necessária para um conjunto de operações essenciais. As operações do primeiro grupo são identificadas como sendo irrelevantes para sempre independentemente das operações que executem no futuro. Assim, qualquer conjunto de operações essenciais não necessitará de incluir estas operações. O quarto grupo inclui operações que não influenciam o estado observável considerando todas as operações executadas – se pudessem ter

impacto, estariam no terceiro grupo. Sendo assim, esta operações não precisam de estar em nenhum conjunto essencial. Todas as outras operações são propagadas para todas as réplicas. Assim, num estado quiescente, cada réplica recebeu todas as operações que podem ter impacto no estado observável.

4.4 Tolerância a falhas

A replicação não uniforme tem como objetivos reduzir os custo de disseminação e o tamanho das réplicas, evitando propagar operações que não influenciem o estado observável do objeto. Esta aproximação levanta questões sobre a durabilidade das operações que não são imediatamente propagadas para todas as réplicas (por serem temporariamente irrelevantes).

Uma solução possível passa simplesmente por propagar todas as operações para pelo menos $f + 1$ réplicas, tolerando assim até f falhas. Deste modo, temos a garantia que uma operação não se perde mesmo tendo f falhas. No entanto, seria necessário adaptar o algoritmo proposto para que no caso em que a réplica recebesse uma operação para propósitos de durabilidade, esta teria de propagar a operação para outras réplicas se a réplica de origem falhasse.

5 CRDTs não uniformes

Nesta secção, mostramos como aplicar o conceito de replicação não uniforme para criar CRDTs baseados em operações [20]. O tipo de dados apresentado é inspirado nos CRDTs computacionais [17], que também permitem que réplicas divirjam num estado quiescente.

5.1 Top-K com remoções

Nesta secção apresentamos um CRDT não uniforme que representa um *top-K*. O tipo de dados permite o acesso aos K maiores elementos adicionados e pode ser utilizado para manter uma tabela de classificação de um jogo online. O algoritmo proposto pode ser adaptado para definir um CRDT que filtre os elementos utilizando uma qualquer função determinística, substituindo a função *topK* utilizada por outro filtro.

A semântica das operações definidas no CRDT é a seguinte: a operação *add(id,val)* adiciona um novo par ao objeto. A operação *rmv(id)* remove todos os pares associados ao identificador que foram adicionados por uma operação que ocorreu antes da remoção. Esta semântica leva a uma política *add-wins*, onde uma remoção não tem impacto em operações de adição concorrentes. A operação *get()* devolve os K maiores pares (de acordo com a ordem total definida para os pares de elementos, usada pela função *topK*).

O algoritmo 2 apresenta um tipo de dados que implementa esta semântica. A operação *prepare-update add* gera um *effect-update* que inclui um parâmetro adicional contendo uma estampilha temporal $\langle replicaid, val \rangle$, onde *val* é um inteiro gerado de forma monotonicamente crescente. A operação *prepare-update rmv*

Algoritmo 2 Top-K com remoções

```
1: elems : set of ⟨id, score, ts⟩ : initial {}
2: removes : map id ↦ vectorClock: initial []
3: vc : vectorClock: initial []
4:
5: GET() : set
6:   return {⟨id, score⟩ : ⟨id, score, ts⟩ ∈ topK(elems)}
7:
8: prepare ADD(id, score)
9:   generate add(id, score, ⟨getReplicaId(), ++ vc[getReplicaId()]⟩)
10:
11: effect ADD(id, score, ts)
12:   if removes[id][ts.siteId] < ts.val then
13:     elems = elems ∪ {⟨id, score, ts⟩}
14:     vc[ts.siteId] = max(vc[ts.siteId], ts.val)
15:
16: prepare RMV(id)
17:   generate rmv(id,vc)
18:
19: effect RMV(id, vc_rmv)
20:   removes[id] = pointwiseMax(removes[id], vc_rmv)
21:   elems = elems \ {⟨id_0, score, ts⟩ ∈ elem : id = id_0 ∧ ts.val < vc_rmv[ts.siteId]}
22:
23: hasImpact(op, S): boolean
24:   R = S • op
25:   return topK(S) ≠ topK(R)
26:
27: maskedForever(log_local, S, log_recv): set of operations
28:   adds = {add(id_1, score_1, ts_1) ∈ log_local :
29:     (∃add(id_2, score_2, ts_2) ∈ log_local : id_1 = id_2 ∧ score_1 < score_2 ∧ ts_1.val < ts_2.val)∨
30:       (∃rmv(id_3, vc_rmv) ∈ (log_recv ∪ log_local) : id_1 = id_3 ∧ ts_1.val < vc_rmv[ts_1.siteId]}
31:   rmvs = {rmv(id_1, vc_1) ∈ log_local :
32:     ∃rmv(id_2, vc_2) ∈ (log_local ∪ log_recv) : id_1 = id_2 ∧ vc_1 < vc_2}
33:   return adds ∪ rmvs
34:
35: mayHaveObservableImpact(log_local, S, log_recv): set of operations
36:   return {}                                   ▷ Este caso não afeta este tipo de dados
37:
38: hasObservableImpact(log_local, S, log_recv): set of operations
39:   adds = {add(id_1, score_1, ts_1) ∈ log_local : ⟨id_1, score_1, ts_1⟩ ∈ topK(S.elems)}
40:   rmvs = {rmv(id_1, vc_1) ∈ log_local :
41:     ∃⟨id_2, score_2, ts_2⟩ ∈ topK(S.elems) : id_1 = id_2 ∧ ts_2.val < vc_1[ts_2.siteId]}
42:   return adds ∪ rmvs
```

gera um *effect-update* que inclui um parâmetro adicional contendo um relógio vetorial que sumariza as operações de adição que ocorreram antes da operação de remoção. O objeto mantém um relógio vetorial que é atualizado sempre que uma nova operação de adição é gerada ou executada localmente. Adicionalmente, este relógio vetorial deve ser atualizado sempre que uma réplica recebe uma mensagem de uma réplica remota (para sumarizar também as operações de adição conhecidas pela origem que não foram propagadas para esta réplica).

Para além deste relógio vetorial, *vc*, cada réplica do objeto mantém: (i) um conjunto, *elems*, contendo os elementos adicionados pelas operações de adição conhecidas localmente (e que ainda não foram removidos); e (ii) um mapa, *removes*, que para cada elemento *id* tem um relógio vetorial que sumariza as operações de adição que aconteceram antes de todas as operação de remoção de *id* (por simplicidade, assumimos que uma chave que não faça parte do mapa tem associado um relógio vetorial contendo apenas zeros para cada réplica).

A execução de uma operação de adição consiste em adicionar o elemento ao conjunto, *elems*, se a operação não aconteceu antes de uma remoção an-

teriormente recebida sobre o mesmo elemento – isto pode acontecer visto que as operações podem não ser propagadas por ordem causal [12]. A execução de uma operação de remoção consiste em atualizar o mapa de remoções, *removes*, e apagar do conjunto, *elems*, a informação relativa às adições do elemento que ocorreram antes da remoção. Para averiguar se uma adição ocorreu antes de uma remoção, verificamos se a estampilha temporal associada à adição está refletida no relógio vetorial da remoção (linhas 12 e 21). Deste modo garantimos a semântica do CRDT, assumindo que as funções utilizadas pelo protocolo são corretas.

Agora analisamos o código dessas funções. A função HASIMPACT verifica se o estado observável é alterado após a execução da nova operação.

A função MASKEDFOREVER calcula: as adições locais que se tornam irrelevantes tendo em conta outras operações de adição (com o mesmo elemento, mas com um valor mais baixo) e remoção (com o mesmo elemento que ocorreram antes da adição); as remoções que se tornaram irrelevantes tendo em conta outras remoções (com o mesmo elemento, mas com um relógio vetorial menor).

A função MAYHAVEOBSERVABLEIMPACT devolve um conjunto vazio, visto que para ter impacto em qualquer estado observável uma operação teria também de ter impacto no estado local observável por si só.

A função HASOBSERVABLEIMPACT calcula: as adições locais que não foram propagadas para outras réplicas e que fazem parte do top-K na réplica local; as remoções locais que removeram um elemento no top-K.

6 Avaliação

Nesta secção comparamos o Top-K proposto neste artigo (NuCRDT) com uma solução que usa replicação total, mantendo o conjunto de elementos em todas as réplicas. Para tal, usamos um CRDT *OR-Set*, que modela um conjunto com suporte para adições e remoções, com uma politica *add-wins* à semelhança do Top-K. A nossa avaliação foi efetuada usando a base de dados chave-valor AntidoteDB [21] e pretende avaliar se o modelo não uniforme permite: (i) reduzir o tamanho das réplicas; (ii) reduzir a quantidade de dados transmitidos; e (iii) aumentar a escalabilidade do sistema.

As experiências foram executadas na plataforma AWS EC2, utilizando instâncias de máquinas do tipo *m3.xlarge* para todos os nós. Todas as experiências usaram 5 nós para o AntidoteDB, cada um a executar nas seguintes regiões: *eu-west*, *eu-central*, *us-east*, *us-west*, e *ap-northeast*. Nas execuções, o tipo de dados foi configurado da seguinte forma: K é 100, os identificadores dos jogadores são selecionados a partir de uma distribuição uniforme tendo um domínio de 10.000, e os valores das pontuações foram selecionados entre 1 e 250.000 usando uma distribuição uniforme. O sistema foi configurado para suportar entre zero a duas falhas, com a réplica de origem a propagar as operações irrelevantes para pelo menos f réplicas.

Como esperamos que a remoção seja uma operação pouco utilizada (usada apenas quando um jogador é removido do jogo), a carga de trabalho é dividida da seguinte forma: 95% para operações de adição e 5% para operações de remoção.

6.1 Tamanho das réplicas e quantidade de dados transmitidos

Para medir o tamanho das réplicas e a quantidade de dados transmitidos entre réplicas utilizamos um único cliente (a correr numa instância EC2 em separado) para executar operações nos 5 nós de AntidoteDB diferentes.

Nesta avaliação o cliente executa uma sequência de operações gerada aleatoriamente sobre os dois tipos de dados a serem comparados. Os valores são obtidos a cada 5.000 operações, obtendo o tamanho total das mensagens enviadas entre cada centro de dados e o tamanho médio do objeto em cada réplica. Os resultados representam o resultado médio de três execuções independentes. A figura 1 mostra os resultados obtidos.

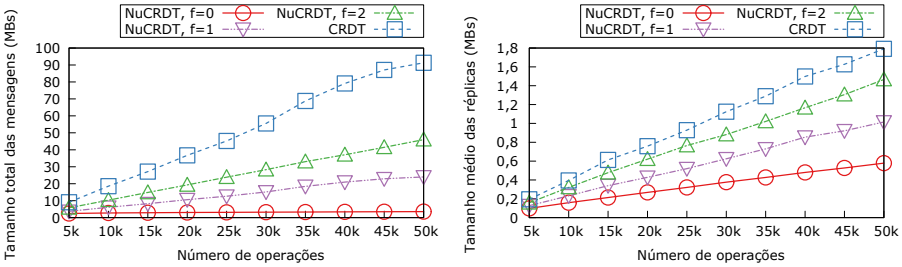


Figura 1. Tamanho total de mensagens e tamanho médio das réplicas do Top-K

O nosso tipo de dados consegue reduzir significativamente o tamanho total das mensagens enviadas entre réplicas (até 96% para $f = 0$) e o tamanho médio das réplicas (até 67% para $f = 0$) quando comparado com o CRDT que usa replicação total. Isto verifica-se principalmente por o CRDT ter de propagar todas operações para todas as réplicas enquanto o NuCRDT propaga apenas as operações essenciais para todas as réplicas e as restantes para um subconjunto de réplicas por questões de durabilidade.

6.2 Escalabilidade

Para medir a escalabilidade dos tipos de dados utilizámos cinco instâncias EC2 extra, cada uma a correr a ferramenta de avaliação Basho Bench [4]. Cada nó Basho Bench executa um número variável de clientes que contactam o nó AntidoteDB a executar na mesma região. Cada execução dura 3 minutos e os resultados apresentados são as médias de três execuções independentes.

A figura 2 apresenta os resultados obtidos, que mostram que ambos os tipos de dados se comportam de forma semelhante com taxas de transferência reduzidas. No entanto, quando se aumenta o número de cliente, a solução com o NuCRDT apresenta melhor escalabilidade, com a solução baseada em CRDTs a ter uma queda drástica da taxa de transferência e um aumento da latência a partir das 4.000 operações por segundo.

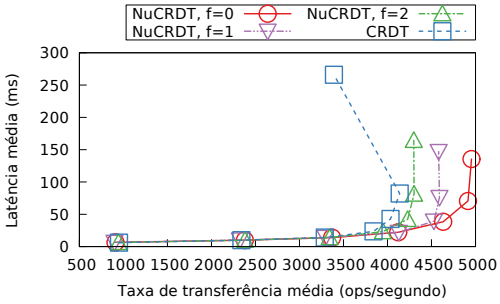


Figura 2. Escalabilidade do Top-K

7 Conclusões

Neste artigo apresentámos o modelo de replicação não uniforme, um modelo alternativo de replicação que combina as vantagens da replicação total, permitindo a qualquer réplica responder a todas as operações de leitura, e da replicação parcial, permitindo que cada réplica mantenha apenas parte dos dados.

Mostrámos como aplicar este modelo à consistência eventual, e propusemos um protocolo de sincronização por troca de operações genérico que fornece replicação não uniforme. Apresentámos também um tipo de dados, o top-K, que utiliza este modelo de replicação não uniforme. As nossas avaliações mostram que a aplicação deste novo modelo de replicação ajuda a reduzir o espaço de armazenamento utilizado, a quantidade de dados transmitidos, e a aumentar a escalabilidade do sistema.

Agradecimentos

Este trabalho foi parcialmente suportado pelo projeto europeu LightKone (grant agreement 732505) e pelo projeto FCT/MCT NOVA-LINCS Ref. UID/CEC/04516/2013.

Referências

1. Almeida, P.S., Shoker, A., Baquero, C.: Efficient state-based crdts by delta-mutation. In: Proc. 3rd International Conference on Networked Systems, NETYS 2015 (2015)
2. Almeida, S., Leitão, J.a., Rodrigues, L.: Chainreaction: A causal+ consistent datastore based on chain replication. In: Proc. 8th ACM European Conference on Computer Systems. EuroSys '13 (2013)
3. Alonso, G.: Partial database replication and group communication primitives. In: Proc. European Research Seminar on Advances in Distributed Systems (1997)
4. Basho Technologies, Inc.: Basho Bench, `https://github.com/basho/basho_bench`
5. Basho Technologies, Inc.: Riak KV, `http://docs.basho.com/riak/kv`
6. Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D., Yerneni, R.: Pnuts: Yahoo!'s hosted data serving platform. Proc. VLDB Endow. 1(2) (Aug 2008)

PART: Árvore ART em Memória Persistente

Godinho Quissico, Daniel Castro, and João Barreto

INESC-ID / Universidade de Lisboa

Resumo As tecnologias emergentes de memória persistente (MP) são candidatas a revolucionar significativamente as arquiteturas de computadores. Recentemente, diferentes propostas têm-se focado nas estruturas de indexação comuns nos sistemas de armazenamento de alta escalabilidade, como por exemplo as árvores B ou B+. Todavia, estas soluções forçam a escrita direta em MP no caminho crítico da aplicação. Deste modo, o desempenho de cada operação que modifique a árvore (inserção ou remoção) é dependente do desempenho da tecnologia de MP utilizada. Este artigo propõe uma abordagem inovadora que, em cenários de aplicações de processamento sequencial de lotes de operações, consegue remover a maioria dos *flushes* de escritas persistentes do caminho crítico. Como prova de conceito, instanciámos a nossa abordagem no caso concreto das *adaptive radix trees* (ART). Os resultados experimentais preliminares confirmam que a nossa abordagem consegue uma melhoria substancial de desempenho até 4.22× mais rápido, face a uma variante baseada em técnicas tradicionais.

1 Introdução

Nos anos recentes, fabricantes de hardware como a Intel, a Micron, a HP, e a Crossbar, têm demonstrado um interesse crescente na criação de memórias persistentes (MP) endereçáveis ao byte [16]. Além de melhorias no consumo energético, estima-se que as MP alcancem latências de leitura comparáveis a atual DRAM, assegurando uma densidade superior. Além disso, as novas MP, sendo persistentes, prometem suportar estruturas de dados que resistam a falhas sem grande impacto no desempenho dos programas que as usam. Assim, as tecnologias de MP são candidatas a produzir uma revolução significativa nas arquiteturas modernas de computadores.

Torna-se então necessário desenvolver soluções que permitam que as aplicações aproveitem o potencial das MP que serão instaladas sobre o *bus* utilizado para a atual DRAM. A alternativa que se revela atualmente mais interessante é permitir que os programas mapeiem regiões persistentes (cujas páginas estão localizadas em MP) no seu espaço de memória virtual [17,18]. Essas regiões persistentes podem, então, ser acedidas diretamente, como se fossem páginas em DRAM.

Esta abordagem é particularmente apelativa pois não implica nenhuma camada de *software* (chamadas de sistema, *caches software*, etc) entre a aplicação e a MP, permitindo assim explorar o real potencial de desempenho da MP. Para este efeito, é necessário assegurar que: *i*) As estruturas de dados mantidas em MP são eficientes tendo em conta as características únicas da MP; nomeadamente, devem minimizar o número de escritas à MP e, para essas escritas, os

flushes da *cache* da CPU; e *ii*) As escritas sobre dados em MP devem sobreviver consistentemente a falhas do sistema (incluindo falhas abruptas da aplicação *runtime*, ou do sistema operativo).

A forma natural de abordar ambos os requisitos é adotar técnicas tradicionais de *logging* (desenhadas para persistir dados de sistemas baseados em blocos, tais como discos magnéticos ou SSD). No entanto, é já consensual que a mera aplicação direta dessas técnicas no contexto de MP não permite explorar o verdadeiro potencial destas novas tecnologias, implicando um número proibitivo de escritas e *flushes* de *cache* da CPU que facilmente degradam o desempenho das estruturas de dados persistentes [1,2].

Recentemente, uma intensa linha de investigação tem proposto técnicas alternativas para persistir estruturas de dados em MP. Estas técnicas conseguem desempenhos muito superiores que os alcançados por técnicas tradicionais. Entre as novas propostas, uma classe importante aborda as estruturas de indexação usadas nos sistemas de armazenamento de alta escalabilidade mais populares atualmente, como por exemplo as árvores B [6,8] ou B+ [1,7,2].

No entanto, o seu desempenho é ainda limitado por dependerem de *flushes* de *cache* que ocorrem no caminho crítico de cada operação que modifica a árvore (e.g., inserção e remoção). Daí, a questão das estruturas persistentes para as MP estar ainda em aberto.

Este artigo propõe a RPVMP, uma abordagem inovadora que, no cenário de aplicações que processam sequencialmente lotes de operações, consegue remover a maioria dos *flushes* de escritas persistentes do caminho crítico das operações que alteram a árvore. Como prova de conceito, instanciámos a nossa abordagem no caso concreto das *adaptive radix trees* (ART). Os resultados experimentais preliminares confirmam que a nossa abordagem consegue uma melhoria substancial de desempenho até 4.22× mais rápido, face a uma variante persistente baseada em técnicas de *logging* tradicionais.

O resto do artigo está organizado da seguinte forma: será feita na Secção 2 a apresentação do modelo do sistema, na Secção 3 a apresentação do algoritmo da variante persistente da ART, na Secção 4 será feita a descrição e avaliação da implementação, de seguida na Secção 5 onde falaremos sobre os trabalhos relacionados e por fim na Secção 6 apresentaremos as conclusões e trabalhos futuros.

2 Modelo do Sistema

Assumimos um sistema com a estrutura de árvore (como veremos na Secção 3.1) que mantém um conjunto dinâmico de elementos, indexados por uma chave. As aplicações que usam este sistema, podem invocar as seguintes operações: *i*) inserção: $\langle node_i, k, v \rangle$, que aloca um espaço em MP para o novo elemento, inicializa-o e por fim insere-o; *ii*) remoção: $\langle node_i, k \rangle$ que faz uma pesquisa da chave no nó pai e efetua a remoção do elemento correspondente; e *iii*) pesquisa: $\langle node_i, k \rangle$, que pesquisa a chave no nó pai e retorna o elemento correspondente. Onde $node_i$ representa o nó pai onde será executada a operação, k o prefixo da chave e v o elemento a ser inserido.

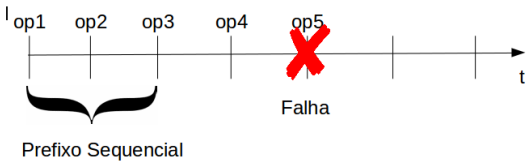


Figura 1: Modelo do Sistema para a abordagem RPVMP

As operações são executadas em lote de forma sequencial. Os resultados das operações que alteram a árvore (inserção e remoção) são visíveis para o exterior apenas no final da execução do lote.

Queremos garantir que, após falha e recuperação, a aplicação observe um estado da árvore que é consistente com a execução de um prefixo (ordenado) do lote que esteja em execução quando a falha ocorreu, como mostrado a Figura 1.

3 Algoritmo da PART na abordagem RPVMP

Nesta Secção, iremos apresentar o algoritmo da abordagem RPVMP para a variante persistente da ART (PART) para aplicações baseadas em lote de operações com execução sequencial sobre MP.

A PART consiste em, sempre que possível, retirar os *flushes* de escritas persistentes do caminho crítico das operações que modificam a árvore (inserção e remoção). Como descreveremos de seguida, tal é conseguido na maioria das operações mais comuns. Intuitivamente, isto é possível combinando a representação de objetos em memória com mecanismos de proteção que o sistema operativo (SO) oferece. Mais precisamente, a PART representa os elementos e os nós intermédios da árvore de forma a que, mesmo que um número de escritas persistentes possa não ter ainda chegado à MP (ou seja, esteja apenas em *cache*) e se perca quando ocorre uma falha, o sistema seja capaz de recuperar um estado que é coerente com um momento anterior à primeira das operações não estáveis que se perdeu.

A propriedade acima permite que cada operação não tenha de esperar por um *flush* das respetivas escritas antes de retornar à aplicação. Em contraste, múltiplas operações do lote podem efetuar as suas escritas apenas em *cache* e retornar de imediato. As respetivas escritas são periodicamente persistidas (caso não tenham ainda chegado a MP) por uma tarefa assíncrona.

Nas restantes secções, começamos por definir a ART (Subsecção 3.1). Depois apresentaremos o algoritmo da PART seguindo uma abordagem gradual: primeiro descrevemos a alocação de novos elementos (Subsecção 3.2); depois complementamos a descrição anterior com a inserção dos novos elementos na árvore (Subsecção 3.4); de seguida, descrevemos as remoções (Subsecção 3.5) e pesquisas (Subsecção 3.6); finalmente, discutiremos sobre operações avançadas (Subsecção 3.7).

3.1 ART

A variante da *Radix-Tree*, *Adaptive Radix Tree* (ART), é uma estrutura de indexação eficiente em memória. Seu desempenho na pesquisa supera as árvores de pesquisa de alto nível, somente de leitura, além de suportar inserções e remoções muito eficientes. Ao mesmo tempo, a ART é muito eficiente em termos de espaço, escolhendo de forma adaptativa estruturas de dados compactas e eficientes para nós internos [3].

Pelo facto de a ART ter uma estrutura de árvore, ela é constituída por nós internos e nós terminais (nós folha). Os nós internos armazenam o par prefixos de chaves e referências e nós folha que armazenam o par chave-valor. A ART utiliza quatro tipos de nós internos diferentes (4, 16, 48 e 256). Estes nós são nomeados segundo a sua capacidade máxima.

A ART utiliza uma representação direta das chaves no lugar de calculo de *hash* ou comparação de chaves (ideia semelhante a ordenação de livros por índices alfabéticos), apresenta uma ocupação espacial global otimizada devido a utilização de nós de tamanho variável. Como forma de reduzir a altura da árvore através da redução de número de nós internos, a ART utiliza duas técnicas bastante eficientes para situações de chaves longas que são: a *lazy expansion* (expansão preguiçosa) e a *path compression* (compressão do caminho).

3.2 Alocação de novos elementos

A PART é implementada sobre um vetor em MP. Os elementos são inseridos na próxima entrada livre utilizando a técnica *append-only*. É utilizado o princípio de páginas de memória virtual do sistema operativo e é feita a ocupação destas de forma sequencial pelo vetor. Entre as páginas, a cada momento identificamos a página que contém a próxima entrada do vetor como a página atual, sendo as páginas anteriores designadas páginas estáveis. Tal como o nome indica, as páginas estáveis contêm entradas que estão "garantidamente"persistentes, enquanto a página atual não oferece essa garantia (algumas entradas podem não ter sido persistidas).

Isto é garantido sem exigir *flushes* de *cache* no caminho crítico das operações. As novas entradas são escritas na página atual e somente quando ela é totalmente preenchida (ou seja, a sua última entrada disponível é escrita), é que todas as escritas efetuadas durante o tempo de vida dessa página são forçadas a persistir em MP. Este procedimento é feito assincronamente por uma tarefa paralela, logo a aplicação pode continuar a realizar operações sobre a nova página atual (i.e., a página seguinte).

O índice que indica qual a página atual é também mantido de forma persistente. Sempre que a página atual é completada e persistida, este índice é incrementado e o novo valor persistido de forma atómica. Todas as operações pertencentes a uma página são armazenadas em um vetor de operações.

Durante a recuperação, o índice é consultado para se determinar qual a página que era a atual aquando da falha. Essa página é invalidada da tabela de páginas do processo através da revogação das permissões de acesso da mesma, anulando assim o sufixo de entradas que tinham sido criadas mas cuja persistência não estava garantida.

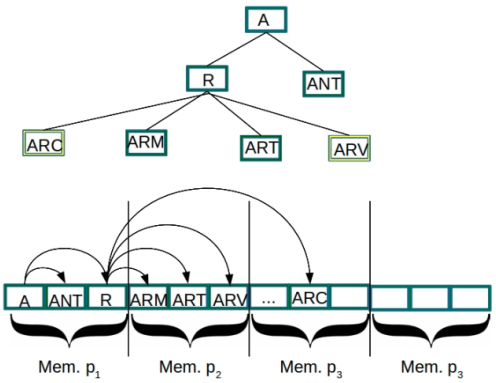


Figura 2: Estrutura lógica e física da PART

Nos casos raros de pesquisas sobre nós de páginas persistentes que tenham eventualmente alguma referência a uma página invalidada, é retornada uma exceção e removida a referência no nó.

3.3 Indexação de novos elementos

As entradas no vetor de objetos sobre MP incluem não só elementos (nós folha) como também um ponteiro raiz, os nós internos (nós pai) e os elementos de remoção (*tombstones*). Numa sequência normal de inserção, os nós ascendentes estarão localizados em entradas anteriores aos seus descendentes (Figura 2).

Designamos instruções a todas as operações que alteram a árvore. A instrução implica alocação de espaço no vetor sobre MP, inicialização e inserção.

No modelo de estrutura da PART, cada nó interno é constituído por um vetor de referências (*children*), um vetor de chaves (*keys*), um vetor de deslocamento (*offsets*), um contador de instruções (*numInstructions*) e um contador de remoções (*numRemotions*), sendo todos os vetores de tamanho igual a capacidade máxima do nó.

3.4 Inserção de elementos

O Algoritmo 1 mostra a instrução de inserção. Para a inserção de uma nova referências diretamente em um nó pai já existente que esteja alocado em página persistente: verificamos se o nó pai tem capacidade para armazenar a nova entrada (linha 2), inserimos a referência na próxima entrada livre (linha 3) do vetor de referências, inserimos a chave na posição correspondente do vetor de chaves (linha 4), inserimos o deslocamento do elemento na posição correspondente do vetor de deslocamento (linha 5), ordenamos o vetor de deslocamento (linha 6) e incrementamos o contador de instruções em um (linha 7). São armazenadas no vetor de operações as escritas (linhas 8-10) para posterior persistência com comandos *flush*.

Algorithm 1 Algoritmo de inserção

```
1: procedure addChild(nodeParent, refParent, key, child)
2:     if numInstructions < capacidade máxima do nó then
3:         nodeParent.children.add(*child)
4:         nodeParent.keys.add(key)
5:         nodeParent.offsets.add(offset)
6:         nodeParent.offsets.sort()
7:         nodeParent.numInstructions++
8:         operations.insert(*child)
9:         operations.insert(*key)
10:        operations.insert(*numInstructions)
11:        return true
12:    if numRemotions > 0 then
13:        new newnode
14:        newnode.copyValidEntrys(nodeParent)
15:        refParent.update(newnode)
16:        return addChild(newnode, refParent, key, child)
17:    caso dos nós 4, 16 e 48
18:    if numRemotions == 0 then
19:        new newnode
20:        newnode.copyValidEntrys(nodeParent)
21:        refParent.update(newnode)
22:        return addChild(newnode, refParent, key, child)
```

3.5 Remoção de elementos

O Algoritmo 2 mostra a instrução de remoção. Para a remoção de referências diretamente em um nó pai já existente que esteja alocado em página persistente: verificamos se o nó pai tem capacidade para armazenar a nova entrada (linha 2), pesquisamos entrada valida no vetor de chaves para a chave pretendida (linha 3), criamos e inserimos o na próxima entrada livre do vetor sobre a MP uma estrutura denominada *tombstone* que representa a remoção de um nó folha (linha 4), inserimos a referência do elemento a ser removido pela *tombstone* (linha 5), inserimos a referência ao *tombstone* no vetor de operações (linha 6) e no vetor de referências do nó pai (linha 7). Por fim, copiamos a chave do nó removido para a posição correspondente da *tombstone* (linha 8), inserimos o deslocamento no vetor de deslocamento (linha 9), ordenamos o vetor de deslocamento (linha 10), incrementamos a um os contadores de instruções e remoções (linhas 14 e 15).

3.6 Pesquisa

O Algoritmo 3 mostra a operação de pesquisa. Dado um nó pai, é efetuada uma pesquisa sobre o vetor de deslocamento utilizando "busca binária"para identificar a ocorrência da chave pretendida (linha 7). De seguida, são percorridas as posições anteriores a posição identificada pela "busca binária"do vetor de deslocamento para contar repetições da chave e armazenada numa variável (linhas 8 a 10). São também percorridas as posições posteriores do vetor deslocamento para contar repetições da chave (linhas 11 a 13). Por fim, são adicionados os

Algorithm 2 Algoritmo de remoção

```
1: procedure removeChild(nodeParent, refParent, key)
2:     if numInstructions < capacidade máxima do nó then
3:         child ← nodeParent.shearchKey(key)
4:         new tombstone
5:         tombstone.ref=child
6:         operations.insert(*tombstone)
7:         nodeParent.children.add(*tombstone)
8:         nodeParent.keys.add(key)
9:         nodeParent.offsets.add(offset)
10:        nodeParent.offsets.sort()
11:        nodeParent.numInstructions++
12:        nodeParent.numRemotions++
13:        operations.insert(*key)
14:        operations.insert(*numInstructions)
15:        operations.insert(*numRemotions)
16:        return true
17:    if numRemotions > 0 then
18:        new newnode
19:        newnode.copyValidEntrys(nodeParent)
20:        refParent.update(newnode)
21:        return removeChild(newnode, refParent, key)
22:    caso particular dos nós 4, 16 e 48
23:    if numRemotions == 0 then
24:        new newnode
25:        refParent.update(newnode)
26:        return removeChild(newnode, refParent, key)
```

Algorithm 3 pesquisa para Nós 16, 48, 256

```
1: procedure findChild(nodeParent, key)
2:     middle
3:     cont
4:     i
5:     low ← 0
6:     high ← numInstructions
7:     nodeParent.offsets.binarySearch(key, low, high, middle)
8:     for <i=middle to 0> do
9:         if nodeParent.keys[i] == key then
10:            cont++
11:    for <i=middle to numInstructions> do
12:        if nodeParent.keys[i] == key then
13:            cont++
14:    if cont == 0 then
15:        return children[middle]
16:    if cont % 2 != 0 then
17:        return children[i]
```

valores. Caso seja zero (chave sem repetições), é retornado o elemento identificado na "busca binária"(linha 15). No caso de número ímpar (última ocorrência valida), é retornado o elemento da última ocorrência na busca sequencial (linha 17). Para os nós do tipo 4, é implementada uma busca sequencial sobre o vetor de deslocamentos para identificação da chave e de suas repetições.

3.7 Operações avançadas

Para operações de expansão preguiçosa e/ou compressão de caminho e substituição de nós (instruções que impliquem a substituição do nó), é utilizada a técnica de *copy-on-write* para garantir a consistência dos dados (melhoria aplicada sobre a versão inicial da PART).

4 Implementação e Avaliação

Nesta Secção, faremos a comparação da PART com o estado da arte (ART) e uma solução trivial que utiliza técnicas de *loggings* tradicionais para garantir consistência dos dados, em termos de desempenho da aplicação e número de *flushes* utilizados no caminho crítico de cada instrução para operações de inserção, remoção e operações de pesquisa nas diferentes tecnologias de MP.

4.1 Implementação

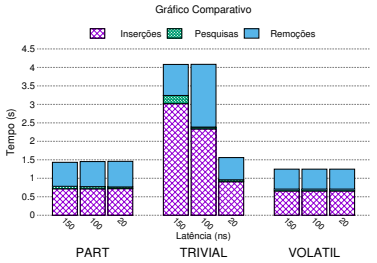
Implementamos a PART com base no código aberto em C da *ART* [23]. A solução trivial foi desenvolvida com base nas técnicas de *logging* tradicionais. Pelo facto da tecnologia de MP não estar ainda disponível, foi feita uma emulação utilizando a biblioteca pmem [22], especificamente as sub-bibliotecas **libpmem** (para o mapeamento da MP) e **libvmem** (para alocação volátil em MP). Para simular as diferentes tecnologias de MP, foram utilizadas três latências de tecnologias proeminentes através da implementação de *spins* com *flush times* de: $2e^{-6}$ para MRAM (20ns), $80e^{-6}$ para RRAM (100ns) e $120e^{-6}$ para PCM (150ns)[4].

4.2 Configuração experimental

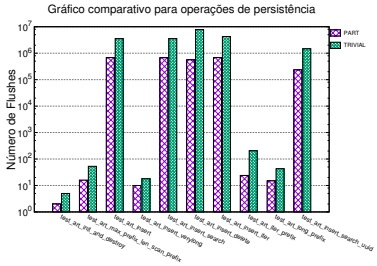
Todas as experiências, foram realizadas num servidor Linux (versão do kernel 4.4.0-81-generic) com um processador Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz, *caches* de 8192 KB e 12GB de memória DRAM. Foram alocados 512MB de memória para emular a MP.

A Figura 3 (a) apresenta os resultados do desempenho das diferentes soluções (ART, trivial e PART) para as três operações (inserção, remoção e pesquisa) utilizando a *benchmark Check* que vem incluída no código da ART e verifica a sua correta implementação.

A PART apresenta ganhos significativos relativamente à solução trivial. A operação de inserção apresenta tempos de resposta de 1.26×, 3.26× e 4.22× mais rápidos, respetivamente, para MRAM, RRAM e PCM. Relativamente à solução volátil, os tempos de resposta são ligeiramente mais elevados, respetivamente, 1.09×, 1.38× e 1.09×, para as diferentes tecnologias.



(a) Comparação de desempenho para diferentes latências



(b) Gráfico comparativo de número de *flushes* no caminho crítico

Figura 3: Desempenho das aplicações para as diferentes tecnologias de MP

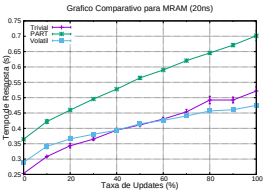


Figura 4: Comparação das três soluções para MRAM ($20ns$)

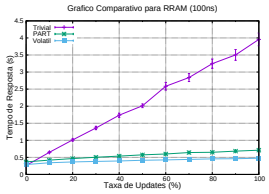


Figura 5: Comparação das três soluções para RRAM ($100ns$)

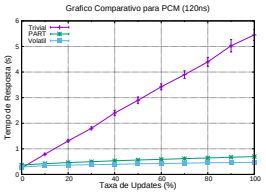


Figura 6: Comparação das três soluções para PCM ($150ns$)

Para a remoção, a PART apresenta ganhos de 1.07×, 2.81× e 2.85× relativamente à solução trivial, e perdas de 1.16×, 1.25× e 1.16× relativamente à solução volátil, respetivamente, para a MRAM, RRAM e PCM.

Na Figura 3 (b), são apresentados os resultados das contagens do número de escritas persistentes no caminho crítico das operações que alteram a árvore utilizando a *benchmark* sintética. Os resultados mostram uma redução em 75% de *flushes* no caminho crítico das operações para a PART em relação à solução trivial.

Para simular cargas de trabalho mais realistas, sintetizamos uma *benchmark* que efetua operações de inserção, remoção e pesquisa de forma aleatória. As Figuras 4, 5 e 6 apresentam os resultados obtidos. Como na anterior, notamos que a PART apresenta ganhos mais evidentes sobre a trivial quanto maior a latência da tecnologia de MP.

5 Trabalho Relacionado

Quando disponível, a MP poderá ser acedida pelas aplicações de maneiras diferentes. A abordagem que possivelmente resultará na menor intervenção nas aplicações existentes consiste em reter as abstrações existentes para o armazenamento em disco, nomeadamente no contexto de sistemas de ficheiros ou base de dados.

Grande parte dos trabalhos iniciais no âmbito da MP, visam reinventar a camada I/O dos sistemas de ficheiros [20] e das bases de dados [4] de forma a beneficiar ao máximo do desempenho e das garantias de consistência que distinguem a MP das restantes tecnologias de armazenamento, ao mesmo tempo que se garante a retro-compatibilidade de aplicações existentes.

Os dados dos aplicativos ainda se encontram separados por um conjunto de objetos voláteis, que são acedidos diretamente através de leituras e escritas de posições de memória, das respetivas cópias persistentes, armazenadas em formato serializado e acessível através de uma interface que interage com o sistema de ficheiros (com possíveis chamadas de sistema). Isto implica uma capacidade de memória considerável e elevados custos no desempenho, nomeadamente na cópia dos dados, serialização e sincronização.

Uma abordagem alternativa expõe a MP como amontoados persistentes onde as aplicações podem ler ou escrever diretamente através de instruções existentes no contexto da memória partilhada (*loads* e *stores*) [17]. Embora esta abordagem seja apelativa, os benefícios de amontoados persistentes dependem da forma de como os dados são acedidos, de tal forma que: *i*) o acesso seja eficiente tendo em conta as características de desempenho que a MP possibilita (ou seja, é fulcral minimizar os altos custos de *flush* da *cache* da CPU e escritas para MP); e, *ii*) as escritas devem sobreviver de forma consistente a falhas onde a execução ou da aplicação ou do SO abortem abruptamente, entrando em um estado de *fail-stop*.

Uma forma de cumprir com estes requisitos é obrigar as aplicações a utilizar estruturas de dados desenhadas para MP. Estas estruturas expõem uma interface que os aplicativos podem utilizar para armazenar os seus dados de forma persistente.

Uma outra abordagem é fornecer uma interface para secções de falhas atómicas sobre amontoados persistentes. Estas secções permitem, de uma forma fácil, que estruturas de dados arbitrárias, quando usadas dentro das regiões delimitadas pelo programador, toleram falhas, assegurando atomicidade. Algumas propostas nesta linha de pensamento concentram-se em secções de falhas atómicas baseadas em bloqueio [21], bem como em transações de memória [17,18] ou árvores B+.

Os esforços mais notáveis nesta linha têm-se centrado em estruturas de índice para grandes quantidades de dados como árvores B [6,8] ou árvores B+ [1,2,7,2]. Estas propostas adotam diferentes técnicas de redução de número de escritas para MP bem como o número de *flushes*, incluindo: permitir escritas desordenadas nos nós [2], persistência seletiva [1,5], atualizações *in-place* [4,9], técnicas de *versioning* [6,7] e técnicas de *logging* [10,11,12,13,15]. Porém, todas elas convergem no facto da necessidade de execução de *flushes* no caminho crítico das operações para garantir a consistência dos dados, o que penaliza o desempenho das aplicações. A PART se distingue das anteriores pelo facto de para quase todas as operações, não necessitar de executar comandos *flush* no caminho crítico das operações, incrementando assim o desempenho da aplicação.

De forma a suportar os elevados custos para escrever de forma atómica em MP, o procedimento para esvaziar as *caches* voláteis da CPU deve ser, tanto quanto possível, executado em paralelo. Esta linha de pensamento é explorada na PART e também no Kamino-Tx [14]. O Kamino-Tx executa as transações especulativamente em memória volátil. Através de um *redo-log*, e uma estrutura de

dados adicional, as alterações são aplicadas posteriormente em MP. Deste modo, as alterações são persistidas fora do caminho crítico de execução da aplicação. A PART distingue-se do Kamino-Tx [19] pelo facto de não necessitar de *logs* ou complexos sistemas de gestão de *logs* em segundo plano e pelo facto de realizar todas suas operações em MP.

A linha de pensamento de utilização de técnicas de *copy-on-write* como forma de evitar escritas persistentes mas ao mesmo tempo garantir a persistência dos dados, é partilhada com a WORT [5].

6 Conclusões e Trabalho Futuro

O problema das estruturas persistentes para as novas MP está ainda em aberto, pois urge encontrar técnicas que permitam remover os *flushes* de escritas persistentes em MP do caminho crítico das aplicações.

Este artigo aborda um cenário importante em sistemas de computação e armazenamento em larga escala: aplicações que processam sequencialmente lotes de operações. Este artigo propõe uma abordagem inovadora que, nesses cenários, consegue remover a maioria dos *flushes* de escritas persistentes do caminho crítico. Como prova de conceito, instanciámos a nossa abordagem no caso concreto das *adaptive radix trees* (ART). Os resultados experimentais preliminares confirmam que a nossa abordagem consegue uma melhoria substancial de desempenho face a uma variante baseada em técnicas tradicionais, graças a uma redução considerável no número de *flushes* no caminho crítico.

Como trabalho futuro, pretendemos abordar a questão da recolha automática de páginas invalidadas devido a falhas. Adicionalmente, planeamos avaliar a solução com *benchmarks* não triviais, tais como o *TPC-C*. Finalmente, pretendemos aplicar a abordagem aqui proposta a outras estruturas de dados persistentes.

7 Agradecimentos

Este trabalho foi financiado por fundos nacionais através da Fundação para a Ciência e a Tecnologia (FCT) com referência UID/CEC/50021/2013. Um especial agradecimento é dirigido a **GARP, C.F. Gama Afonso Despachante Oficial** (Moçambique) pelo apoio dado para a realização do trabalho.

Referências

1. Yang, J., Wei, Q., Chen, C., Wang, C., Yong, K. L., & Clara, S.: NV-Tree: Reducing Consistency Cost for NVM-based Single Level Systems FAST'15 Proceedings of the 13th USENIX Conference on File and Storage Technologies, 167–181 (2015).
2. Chen, S., & Jin, Q.: wB-tree: Persistent B+-Trees in Non-Volatile Main Memory. VLDB'15, 8(7), 786–797 (2015).
3. Leis, V.,Kemper, A., Neumann, T.: The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases ICDE, 2013.
4. Arulraj, J., Pavlo, A., Dulloor, S.: Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

Monitorização de Sistemas Tolerantes a Faltas Bizantinas para Suportar Adaptação Dinâmica

Bernardo Palma, Daniel Porto and Luís Rodrigues

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
LASIGE, Faculdade de Ciências, Universidade de Lisboa
{bernardo.palma, danielporto, ler}@tecnico.ulisboa.pt

Resumo Os sistemas adaptáveis (SA) mudam o seu comportamento em resposta a variações no ambiente de execução causadas, por exemplo, por faltas ou alterações nos padrões de acesso. Um componente importante em qualquer SA é o sistema de monitorização (SM), responsável por recolher informações sobre a execução e detetar mudanças que justificam a adaptação. O SM é especialmente complexo na presença de faltas Bizantinas, onde elementos do sistema podem produzir mensagens incorretas, que por sua vez podem disparar adaptações indesejáveis, tornando o SA ineficiente ou vulnerável. Neste trabalho, descrevemos i) as escolhas realizadas na construção de um SM robusto e flexível para gerir vários tipos de sensores; ii) os mecanismos de consenso que garantem a obtenção de uma vista coerente do estado do sistema, agregando de forma tolerante a faltas as informações recolhidas pelos sensores. A avaliação demonstra que o nosso SM é facilmente extensível e tem capacidade de escala.

1 Introdução

Os sistemas adaptáveis modificam o seu comportamento em resposta a mudanças nas condições do ambiente de execução que podem ser causadas, por exemplo, por faltas, variações no padrões de acesso durante o dia, utilização de recursos partilhados, carga no sistema imposta pelo número de utilizadores, etc. Estas mudanças dinâmicas provocam alterações que afetam o desempenho dos diferentes componentes do sistema. Por exemplo, alguns protocolos de replicação tolerantes a faltas Bizantinas (TFB) como o Zyzzyva [1] operam sobre condições favoráveis, num modo designado por otimista, onde apresentam o seu melhor desempenho. Entretanto, na presença de faltas, o Zyzzyva precisa executar complexas etapas adicionais de comunicação, que acarretam um maior número de mensagens, reduzindo o seu desempenho [2]. De igual forma, os outros protocolos TFB que se encontram na literatura estão também otimizados para condições específicas de operação [3,4,5,6,7].

A ausência de uma solução padrão, isto é, adequada a todas as condições de operação, criou a oportunidade para construção de sistemas TFB adaptáveis que alternam entre diferentes protocolos em resposta às mudanças dinâmicas no ambiente de execução [6,8,9].

Um componente importante em qualquer sistema adaptável é o sistema de monitorização (SM), que é responsável por recolher informações sobre o funcionamento do sistema, informações estas que podem ser usadas para alimentar as políticas de adaptação. Este componente é particularmente complexo na presença de faltas Bizantinas, onde as réplicas incorretas podem produzir mensagens para ativar adaptações inadequadas, tornando o sistema mais vulnerável a ataques ou passível de apresentar um desempenho subótimo. Para evitar estes problemas, o sistema de monitorização deve também ser resistente a falhas Bizantinas. Isto implica ser capaz de tolerar para além de réplicas incorretas, tolerar também sensores incorretos. Isto é, mesmo que uma fração f dos sensores produzam valores errados, o SM deve conseguir alimentar as políticas com informações exatas e coerentes sobre sistema alvo. Note-se que mesmo as réplicas corretas de um mesmo sensor podem naturalmente divergir nos valores fornecidos. Por exemplo, as leituras podem estar levemente desfasadas no tempo, fazendo com que sensores corretos obtenham leituras com valores distintos. Para além destes desafios, é importante que sistema de monitorização tenha uma arquitetura flexível para permitir o desenvolvimento de novos sensores.

Neste trabalho apresentamos MonBiz, um SM autónomo, robusto e flexível que permite a criação e gestão de diversos tipos de sensores, agrupando informações de diversas fontes para exportar uma visão coerente do estado do sistema, mesmo na presença de faltas Bizantinas. O remanescente deste artigo está organizado da seguinte forma: na Secção 2, apresentamos o contexto em que se insere o nosso trabalho, com foco particular na estratégia de monitorização dos protocolos adaptáveis. Na Secção 3 apresentamos o modelo do sistema e as premissas que temos em consideração. Na Secção 4 descrevemos a visão geral, detalhando a arquitetura, interfaces e principais funcionalidades bem como as escolhas realizadas no desenho do sistema. Na Secção 5, identificamos os parâmetros quantitativos e qualitativos e mostramos os resultados da avaliação do sistema. Por fim, as conclusões são apresentadas na Secção 6.

2 Trabalho Relacionado

Existem na literatura diversos sistemas de monitorização desenvolvidos especificamente para detetar falhas, como o Falcon[10], Pigeon[11] e Albatross[12]. Estes sistemas foram desenvolvidos com um modelo de falhas por paragem, e portanto, não são adequados para um ambiente suscetível a faltas Bizantinas. Naturalmente, possuem similaridades com o MonBiz. Por exemplo, o Falcon disponibiliza uma API para desenvolvimento de sensores, mas diferentemente do MonBiz, pode terminar processos para garantir coerência das informações. O Pigeon é capaz de detetar também falhas na rede e ao contrário do Falcon não mata processos, porém não é capaz de garantir precisão da informação quando a falha é detetada. O Albatross opera em ambientes em que existe um gestor de SDN, ele também atua para garantir coerência das informações mas diferentemente do Falcon, desconecta processos ao invés de mata-los. Otimizações propostas nestes sistemas poderão ser exploradas em futuras versões do MonBiz.

Detetor de faltas Bizantinas (DFBs) [13] é uma abstração que permite construir protocolos TFB a partir de elementos que dão indicações não confiáveis sobre faltas em processos que, respeitando algumas premissas, podem resolver o consenso. Um tipo de DFB proposto em [14] para redes sem-fio, usa a ordem lógica em que os processos entregam mensagens, observando o padrão de comunicação durante a execução do protocolo para detetar faltas. O nosso sistema não propõe novas primitivas para resolver o consenso. Ao invés disto, utiliza o consenso para estabelecer uma visão coerente de diversos sensores para, além de detetar falhas, monitorizar alterações no ambiente de execução que justifiquem uma adaptação.

Alguns protocolos adaptáveis TFB como o *Aliph*[6], ADAPT[8] e *Bytam*[9], têm no seu núcleo um componente de monitorização, responsável por detetar e notificar falhas ou eventos importantes em componentes do sistema ou na rede, e desta forma despoletar adaptações.

O *Aliph* é um dos primeiros protocolos TFB capaz de suportar adaptação dinâmica. Em essência, o sistema alterna por uma ordem fixa entre três protocolos diferentes, nomeadamente *Quorum*, *Chain* e *PBFT*. A troca entre estes protocolos é definida por condições definidas de forma estática no código, que dependem da operação do protocolo em execução, não existindo um sistema de monitorização explicita do ambiente de execução.

O ADAPT já recorre a um sistema de monitorização, designado por Sistema de Eventos (SE), para alimentar técnicas de aprendizagem automática que são, por sua vez, responsáveis por escolher, em cada momento, a melhor configuração para o sistema. A concretização do SE é bastante simples, limitando-se a ler informações da aplicação para obter parâmetros, tais como o tamanho das mensagens e o número de clientes, não concretizando nenhuma técnica de tolerância a faltas (e como tal, não sendo robusto na presença de sensores Bizantinos).

O Bytam, apresenta maior flexibilidade comparativamente ao *Aliph* e ao ADAPT para o desenvolvimento de políticas de adaptação, permitindo defini-las de maneira mais geral, na forma de *Evento-Condição-Ação*. Além disto, contrastando com o ADAPT, o Bytam possui um conjunto de monitores concretizados de maneira robusta, sendo capaz de tolerar faltas Bizantinas. No entanto, a infraestrutura sensores que compõe o módulo de monitorização ainda é demasiado rígida, restringido-se a um conjunto fixo de sensores desenvolvidos de forma integrada ao sistema, não sendo simples estender para adicionar novos sensores.

O MonBiz tem por objetivo superar estas limitações, fornecendo uma infraestrutura de monitores flexível, que permite comunicação com sensores simples e replicados; extensível, tornando mais simples a adição de novos sensores através da implementação de uma API; e robusta sendo capaz de tolerar falhas no agregador e sensores replicados.

3 Modelo do Sistema

Assumimos um sistema distribuído composto por vários processos que comunicam por troca de mensagens. O sistema é assíncrono, no sentido em que

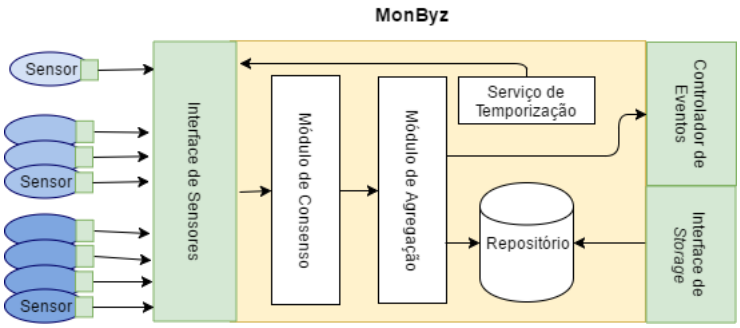


Figura 1. Arquitetura do sistema

os tempos de processamento e de comunicação podem ocasionalmente exceder qualquer limite que se tente impor previamente. No entanto, assumimos que uma maioria dos processos, incluindo aqueles que executam sensores, tem um comportamento no tempo que não inviabiliza a adaptação atempada do sistema. Os processos têm também acesso a uma fonte externa de tempo. Esta fonte é usada pelos sensores para estampar as leituras com uma marca temporal; assumimos que a sincronização dos relógios é externa ao nosso sistema e é realizada por uma fonte confiável.

Os diversos componentes do sistema estão sujeitos a faltas Bizantinas, podendo exibir comportamentos arbitrários. Estas faltas podem ter causas naturais ou resultar da intrusão de agentes maliciosos. Assumimos que no máximo $f = \lfloor \frac{n+1}{3} \rfloor$ réplicas de cada componente podem exibir um comportamento Bizantino, em que n é o número total de réplicas; estas réplicas podem entrar em conluio para tentar subverter o sistema. Assumimos que os processos possuem recursos limitados e não conseguem quebrar as técnicas de criptografia utilizadas nos nossos algoritmos.

4 Arquitetura

O sistema de monitorização MonBiz é composto por três tipos de componentes principais apresentadas na Fig. 1, nomeadamente: um *Agregador*, um *Repositório*, um conjunto de *Sensores*, e um serviço de temporização.

Os sensores recolhem informações sobre o sistema a adaptar (SA) e sobre o ambiente de execução. Os sensores suportam dois modos distintos de operação: podem reagir a eventos esporádicos, por exemplo, falhas, ou fornecem um fluxo de atualizações sobre alguma característica do SA, por exemplo, fornecer leituras periódicas da carga do sistema. Para monitorizar um sistema de forma tolerante a faltas é natural a utilização de múltiplos sensores, cujos valores podem ser comparados e/ou agregados, para desta forma obter maior robustez. Consequentemente podem ser geradas notificações replicadas para um mesmo evento, provenientes de diversos sensores. Um sensor com defeito pode reportar

Tabela 1. Propriedades do sensores

Nome	identifica sensores simples ou grupos de sensores (réplicas) de forma única
Identificador	distingue entre as diferentes réplicas de um sensor (quando aplicável)
Cardinalidade	tamanho do grupo sensores
Faltas	limite de falhas com qual o sensor (ou grupo) consegue operar
Modo de operação	reativo ou periódico
Tipo	Bizantino, *crash*, não-confiável
Credenciais	chaves para autenticar a origem dos valores transmitidos

eventos que não ocorreram, omitir os que aconteceram ou fornecer uma descrição errada sobre os mesmos.

Para além disso, assumimos que o sistema gerido é tolerante a faltas e, como tal, possui várias réplicas que precisam de ser monitorizadas. Cada sensor pode monitorizar uma ou mais réplicas, e cada réplica deve ser monitorizada por pelo menos um sensor.

O MonBiz recolhe os dados de todos os sensores num serviço *Agregador*, que usa os valores fornecidos por diferentes réplicas para filtrar os dados incoerentes produzidos pelos sensores defeituosos. Finalmente, os dados já processados são armazenados de maneira coerente e robusta num *Repositório*, para permitir a consulta por parte de outros serviços. Uma descrição pormenorizada deste componentes será fornecida nas próximas secções.

4.1 Sensores

Os sensores têm a função de recolher informações do SM e do ambiente de execução. O MonBiz permite configurar um sensor para tolerar diferentes tipos de faltas, nomeadamente é possível configurar um sensor como não tolerante a faltas, tolerante a faltas por paragem ou tolerante a faltas Bizantinas. O tipo de configuração escolhida depende de vários fatores, como a semântica do valor a recolher, da possibilidade de instalar ou não sensores independentes para observar um dado fenómeno. Por exemplo, o débito do sistema ou número de clientes ativos podem ser observados por diferentes réplicas e, como tal, ser lidos por sensores independentes, o que permite ter um sensor tolerante a faltas Bizantinas para estas métricas. Por outro lado, alguns sensores possuem características que não permitem a sua replicação ou cuja replicação não faz sentido (por exemplo, estatísticas da CPU de um determinado servidor).

No MonBiz os sensores recolhem dados do sistema através de sondas, de forma passiva, ou podem ainda encapsular sensores legados, enviado esse mesmos dados ao *Agregador*. Desta forma assemelham-se a clientes que enviam comandos para a máquina de estados replicada (MER). Entretanto, não precisam de esperar por uma resposta do *Agregador*, visto que sensores não executam ações sob o sistema. Esta restrição é intencional, visto que adaptações eficientes requerem conhecimento das estratégias do SA, e por isto entendemos que devem ser sistematizadas e coordenadas por um serviço específico. As propriedades dos sensores no MonBiz, são descritas na Tabela 1.

Cada sensor envia ao serviço Agregador uma mensagem contendo a informação recolhida associada a um número de sequência atribuído localmente e incremen-

Listagem 1.1. Classe PeriodicSensor

```
1 package argus.sensors;
2 ...
3 public abstract class PeriodicSensor extends BaseSensor{
4 public PeriodicSensor(Integer id, String type, PrivateKey pKey){...}
5 public abstract SignedMessage collectValue();
6 }
```

Listagem 1.2. Exemplo de configuração do Agregador

```
1 identifier=Throughput
2 sensorType=Metric
3 f=1
4 quorum=3
5 aggregationFunction=argus.aggregator.function.IntAverageFunction
```

tado a cada nova mensagem. Para além disto, a mensagem também deve conter o nome e identificador do sensor e um registo de tempo indicando quando a leitura foi feita. Desta forma, sensores replicados podem ser agrupados no Agregador como um único sensor virtual. No caso de sensores tolerantes a falhas Bizantinas, a mensagem enviada ao Agregador deve ser assinada por cada sensor.

Classificamos a informação recolhida em dois tipos: métricas e eventos. Definimos as métricas como valores numéricos que representam a informação recolhida (e.g., a carga média da CPU de uma dada réplica). Por outro lado, classificamos como eventos, a ocorrência de situações como mudança de líder ou a deteção de falha em uma réplica.

À medida que os sistemas evoluem, torna-se necessário monitorizar um conjunto adicional ou diferente de parâmetros. Desta forma, o MonBiz deve ser capaz de expandir sua capacidade de monitorização. De forma a facilitar o desenvolvimento de sensores, disponibilizamos algumas interfaces em que, por exemplo, apenas é necessário implementar o método de coleção, e.g Listagem 1.1. Para além disto, adaptações como recolocação ou troca de réplicas, quando realizadas, podem desligar ou instalar sensores em conjunto, e como resultado possivelmente mudar a configuração do grupo de sensores. Portanto, o Agregador disponibiliza uma interface que permite o gerência de novos sensores, para desconectar antigos, adicionar e remover réplicas, de maneira atómica e integrada mudar as funções de agregação aplicadas aos sensores. No momento da inicialização o sistema, também permite registar/carregar automaticamente as informações referentes aos sensores existentes, bastando defini-las num ficheiro chamado *aggregation.config* (e.g Listagem 1.2) para cada sensor, juntamente com uma diretoria contendo as chaves públicas pertencentes ao mesmo.

4.2 Agregador

Para simplificar a implementação dos sensores e conferir robustez, o *Agregador* foi desenvolvido como um serviço replicado, estendendo uma biblioteca

para implementação de máquinas de estados replicada tolerante a falhas bizantinas bastante popular denominada BFTSMaRt[15]. Assim, os valores recolhidos através de mensagens dos sensores são ordenados pelo protocolo TFB subjacente. No final do mesmo, todas as réplicas corretas do agregador concordam com a ordem e o conjunto de valores recolhidos dos sensores distintos. Uma vez recolhida uma quantidade mínima de mensagens e estabelecida uma visão coerente dos valores recebidos no agregador, é aplicada uma função de agregação para uniformizar os valores recebidos.

Cada sensor replicado possui uma função de agregação determinista que é aplicada quando a visão dos sensores sobre um determinado evento atinge consenso. Isto permite ao agregador eliminar valores extremos, convergindo a visão dos sensores num único representante da métrica/evento, bem como decidir o respetivo instante associado. Esta função é aplicada pelo módulo de Agregação apresentado na Fig. 1. Um exemplo simples desta função de agregação é computar uma média eliminando as valores mais baixos e mais altos de f sensores, ou ordena-los e selecionar o valor central. Uma vez que o protocolo TFB garante que cada réplica correta do agregador decide pelos mesmos valores obtidos pelos sensores na mesma ordem e a função de agregação é determinista, todas as réplicas corretas obtém o mesmo valor agregado.

Embora não seja estritamente necessário agregar dados recolhidos por sensores não-replicados, a função de agregação pode servir para realizar um pré-processamento do valor recebido, por exemplo, aumentando a sensibilidade do sensor multiplicando o valor recebido por um fator, ou agrupar valores de sensores com frequência mais alta para aliviar o custo no processamento.

4.3 Repositório

Embora as adaptações possam ser iniciadas em resposta a eventos imediatos ou recentes, alguns sistemas realizam adaptações pro-ativas, com base em previsões do comportamento do sistema a partir da observação de acontecimentos ao longo do tempo. Para permitir este tipo de adaptação, o MonBiz regista os dados recolhidos pelos sensores num repositório e disponibiliza os mesmos, já reconciliados, através da interface de *armazenamento*, que pode ser acedida por outros sistemas, como por exemplo um gestor de políticas. O protótipo do serviço de repositório foi concretizado utilizando o H2 [16], uma base de dados relacional. Para garantir a coerência das consultas ao repositório por diferentes réplicas do gestor da adaptação, cada entrada é marcada com um identificador numérico único. Este funciona como o número de versão do repositório, podendo ser usado para limitar a consulta até uma determinada versão/entrada.

4.4 Controlador de Eventos

Apesar de também serem guardados no repositório, eventos são um tipo de informação que por norma requer uma reação imediata. Como tal, aquando da sua ocorrência o sistema alimentado pelo MonBiz poderá querer despoletar uma ação relacionada com o mesmo. Para isso, o nosso sistema oferece a possibilidade

de registar *handlers* associados a determinados eventos. Quando o evento é capturado pelos sensores e devidamente agregado, o respetivo *handler* é chamado podendo, por exemplo, meramente informar o outro sistema da sua ocorrência.

4.5 Serviço de Temporização

O MonBiz oferece também um simples serviço de temporização que pode ser útil para notificar, por exemplo, um gestor de adaptação que um certo período de carência terá terminado ou que estará na altura de reavaliar as suas políticas. Porém, esta notificação só acontece quando o temporizador for despoletado numa maioria das replicas do sistema e passado pelo consenso do mesmo. Os temporizadores funcionam como clientes da maquina de estados replicada, usando um cliente interno ao MonBiz. Estes enviam os eventos à medida que são despoletados, e tal como acontece com os sensores, estes valores passam pelo agregador onde uma acumulação acontece. Finalmente quando o *quorum* é atingido, o evento pode então ser entregue. O serviço disponibiliza dois tipos de temporizadores: Temporizadores periódicos (que, como o nome indica, são despoletados repetidamente com um período de tempo definido no momento do seu registo) e temporizadores que apenas são despoletados uma vez, com um atraso também ele definido no momento do seu registo.

4.6 Módulo de Consenso

Conforme mencionado, o agregador agrupa os valores dos sensores para garantir que todas as réplicas corretas executam operações sobre a mesma visão coerente dos dados. As mensagens de sensores são relacionadas de acordo com as propriedades especificadas para o sensor ou grupo de sensores replicados, definidas no agregador. De maneira semelhante, é definido o limite mínimo para a quantidade de mensagens que cada tipo de sensor deve obter com base na quantidade de faltas toleradas, cardinalidade e tipo de sensor, mostrado na secção 4.1. As mensagens dos sensores podem ser ordenadas à medida que chegam ou acumuladas para posterior ordenação. Experimentámos três abordagens para concretizar este módulo e reportamos o seu comportamento na Secção **??**.

Na primeira concretização, denominada CPré, cada valor enviado pelos sensores é totalmente ordenado, através do consenso. Depois é entregue ao agregador para acumulação e eventual agregação após ser atingido o *quorum* definido para o respetivo sensor. Isto significa que mesmo para sensores replicados, um consenso é corrido para cada valor recebido (ignorando possível *batching*). Aquando da entrega do valor ordenado ao agregador, a validade do mesmo é testada pela seguinte ordem: primeiro, verifica-se que existe um sensor com o mesmo nome do valor recebido registado no sistema; depois, valida-se a autenticidade do valor, verificando que foi corretamente assinado pelo sensor correspondente ou uma da suas réplicas; posteriormente, verifica-se a frescura do valor, nomeadamente se o número de sequência associado ainda não foi acumulado; por último verifica-se se o *quorum* foi atingido.

As duas restantes concretizações, CPós-Total e CPós-Dispersa, executam a acumulação antes da ordenação. Assim sendo, nestas os sensores enviam os seus valores diretamente para o agregador para serem acumulados, sem ordem acordada. Durante esta acumulação, cada um dos valores passa pela verificação descrita acima. Quando o *quorum* é atingido, a acumulação é enviada por um cliente interno para ser totalmente ordenado, sendo finalmente corrido o consenso. Esta escolha de concretização adiciona, portanto, um passo extra de comunicação ao protocolo. Após o conjunto de valores ser ordenado é entregue novamente ao agregador para poder finalmente ocorrer a agregação, passando primeiro por um passo de validação semelhante ao anterior.

Na variante CPós-Total, todas as réplicas corretas enviam o conjunto de valores acumulados assim que atingem o *quorum* para serem totalmente ordenados. Como tal, as acumulações são efetivamente replicadas, não sendo necessário uma réplica líder. Assim que uma destas mensagens passa na verificação final, após a ordenação, as restantes são descartadas. Isto significa que o número de consensos que precisa de ser executado não difere do CPré, apresentando assim um custo claro. Mesmo no melhor caso, esta desperdiça trabalho feito, devido à replicação das mensagens enviadas para o consenso mas, por outro lado, mostra uma maneira simples de concretizar acumulação pré-consenso, sem alterar a máquina de estados replicados e sem necessitar de explicitamente tratar de possíveis falhas ocorridas.

A variante CPós-Dispersa tenta melhorar a anterior. Nomeadamente tenta evitar correr um consenso para cada valor recebido. Para isso, o envio das acumulação é distribuído entre as réplicas, aplicando uma função de *hash* ao nome único do sensor. Assim, cada um dos sensores é atribuído a uma determinada réplica. No caso ótimo, sem a ocorrência de falhas bizantinas, cada réplica só se encarrega de enviar as acumulações dos sensores que lhe compete. Com esta otimização a carga é, portanto, distribuída entre as réplicas. Porém, existe a possibilidade da propriedade de progresso de determinados sensores replicados ser afetada quando uma das réplicas se torna bizantina, quer por enviar valores errados quer por não os enviar. Para lidar com estes casos, quando as réplicas corretas detetam falta de progresso com um determinado sensor, revertem para a primeira concretização, CPós-Total, para os sensores atribuídos à réplica bizantina.

5 Avaliação

A avaliação apresenta uma análise comparativa do desempenho das diferentes concretizações propostas. Uma discussão das qualidades da moldura proposta, nomeadamente da facilidade com que permite desenvolver novos sensores, é omitida por falta de espaço, podendo ser encontrada em [17].

As réplicas do sistema e sensores foram lançados em ambientes virtualizados hospedados pelo serviço *DigitalOcean*, cada um lançado no seu ambiente individual. As máquinas virtual tem acesso a 4 cores de CPU virtualizados, 8GBs de RAM e com um disco SSD de 80GBs cada. Tendo em conta que o nosso sis-

tema foi desenvolvido com recurso à biblioteca Bft-SMaRT que foi desenvolvida em Java, utilizamos a versão 1.8.0_131 para correr o nosso sistema, mantendo os limites por omissão da *heap*. De forma a avaliar o desempenho das diferentes concretizações, estas foram sujeitas a diferentes níveis de carga. Nas experiências, um valor é considerado decidido a seguir a ser agregado, como tal, de forma a que os resultados não sejam afetados pela complexidade da função de agregação, utilizamos uma que apenas retorna o primeiro valor do conjunto obtido. Para além do mais, os valores decididos não são guardados no repositório para testar apenas a capacidade do algoritmo. Tendo em conta que queremos analisar os limites de cada concretização com acumulação prévia antes do consenso e compara-la com a implementação base em que a acumulação é feita posteriormente ao consenso, os testes serão sintéticos na carga que vão gerar, não representando portanto cenários de uso reais. Na instalação do agregador, definimos $f = 1$ gerando, portanto, 4 réplicas do mesmo.

O sensor utilizado é baseado no *microbenchmark* já oferecido pela biblioteca de MER. Cada sensor é lançado como um fio de execução independente que envia sempre o mesmo valor corretamente assinado, sem que ocorra nenhuma espera de forma a atingir um nível elevado de carga. Este valor é enviado um total de 10000 vezes por cada sensor. Nesta avaliação, alteramos a quantidade dos sensores que são lançados e alternamos entre replicados e não replicados. Nos testes com sensores não replicados em cada máquina cliente são lançado 16 sensores em simultâneo, enquanto que nos replicados são apenas lançados 4 sensores. Note-se porém que cada um destes últimos são na realidade 4 fios de execução diferentes, gerando portanto um nível de carga equivalente entre os dois tipos de maquinas cliente. Em ambos os testes, com sensores não replicados e replicados, executamos seis experiências cada, aumentando o número de máquinas cliente de 1 até 6, perfazendo então 16, 32, 48, 64, 80, 96 e 4, 8, 12, 16, 20, 24 sensores para os testes não replicados e replicados, respetivamente. Os valores de débito foram obtidos fazendo a mediana dos valores máximos atingidos entre as 4 réplicas. Para além do mais, é importante referir que a função de dispersão da concretização CPós-Dispersa distribui os sensores de forma uniforme entre as diferentes réplicas.

Os resultados das nossas experiências com sensores não replicados são apresentados na Figura 2(a). Como esperado, a variante CPós-Total apenas conseguiu manter um desempenho semelhante às restantes variantes nas duas primeiras experiências, começando o seu débito a cair daí em diante. Salienta-se também que a partir da segunda experiência, inclusive, não foi capaz de concluir as mesmas, ocorrendo sempre uma exceção por esgotar a memória *heap* do Java. Isto acontece devido à ineficiência desta implementação, visto gerar uma quantidade de mensagens superior às restantes, bem como apresentar uma maior carga de trabalho, em parte devido também ao número de mensagens. Em relação às restantes, podemos verificar que atingem desempenhos semelhantes na maioria das experiências. Apenas na última se nota diferença considerável em que a variante CPós-Dispersa aparenta ter atingido um possível máximo, e a variante CPré mostra conseguir processar cerca de 200 operações a mais.

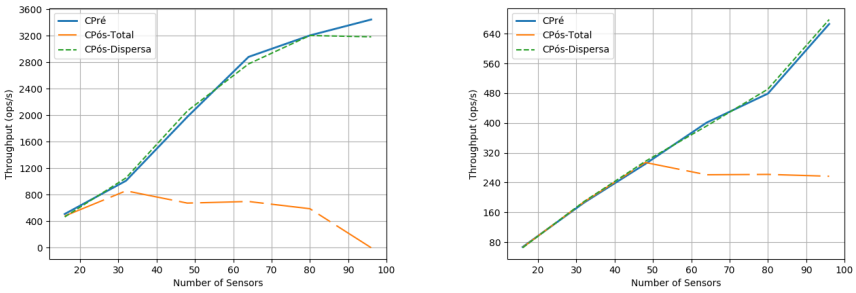


Figura 2. Desempenho das três variantes, sem e com replicação dos sensores.

Nos resultados com sensores replicados, apresentados na Figura 2(b) e salientando novamente que quatro sensores equivalem a um replicado, o comportamento repete-se, no sentido que a variante CPós-Total mais uma vez apresenta o pior desempenho. Porém desta vez podemos ver que a sua capacidade de lidar com as mensagens dos sensores apenas começa a cair a partir dos 12 sensores replicados que corresponde na realidade a uma carga de mensagens equivalente a 48 sensores não replicados. Este mais uma vez não consegue completar todas as experiências, neste caso as últimas três. Com as variantes CPré e CPós-Dispersa não existem disparidades significantes no desempenho para as cargas testadas, como se pode constatar.

De forma geral, conseguimos ver que as variantes CPré e CPós-Dispersa mostram resultados coerentes e comparáveis, apesar da primeira conseguir obter melhor desempenho com sensores não replicados. Isto significa que existe algum mérito na variante CPós-Dispersa, permitindo-nos também argumentar que uma solução mais integrada, nomeadamente entre o agregador e a biblioteca Bft-SMaRT poderia ter um desempenho semelhante ou superior.

6 Conclusão

Neste artigo apresentámos a arquitetura de um sistema de monitorização tolerante a falhas bizantinas com o objetivo de alimentar um gestor de adaptação. Para além disso apresentamos também algumas das interfaces que permitem estender a infraestrutura de sensores do sistema. Tanto quando é do nosso conhecimento, este é o único sistema de monitorização tolerante a falhas bizantinas e que considera sensores com diferentes modelos de falha, incluindo faltas por paragem e Bizantinas.

Agradecimentos: Este trabalho for parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projectos com referências PTDC/ EEI-SCR/ 1741/ 2014 (Abyss) e UID/ CEC/ 50021/ 2013.

Adaptação Dinâmica de Protocolos de Consenso Bizantino

Carlos Carvalho, Daniel Porto, Luís Rodrigues, and Alysson Bessani

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{carlosacarvalho, danielporto, ler}@tecnico.ulisboa.pt
LASIGE, Faculdade de Ciências, Universidade de Lisboa
anbessani@ciencias.ulisboa.pt

Resumo O problema do consenso distribuído na presença de faltas bizantinas tem recebido particular atenção nas últimas décadas. Existem hoje diversos protocolos para este efeito, cada um otimizado para condições de execução particulares. Uma vez que na maioria dos casos os sistemas reais operam em condições dinâmicas, importa desenvolver mecanismos que permitam adaptar os protocolos em tempo de execução ou substituir um protocolo por outro mais adequado às condições correntes. O problema da adaptação dinâmica de protocolos de consenso não é novo, mas a literatura é escassa para o caso bizantino e não existem trabalhos que permitam comparar as soluções existentes. Este trabalho tem dois objetivos complementares. Em primeiro lugar, estuda como as diferentes técnicas de adaptação dinâmica propostas para o modelo de falta por paragem podem ser aplicadas na presença de faltas bizantinas. Em segundo lugar, através da concretização destas técnicas numa moldura de software comum, baseada no pacote de código aberto BFT-SMaRt, apresenta um estudo comparativo do desempenho das mesmas.

1 Introdução

A replicação de máquinas de estado (RME)[19] é uma das técnicas fundamentais para fornecer tolerância a faltas. No seu núcleo, esta técnica pressupõe a execução de um protocolo de consenso distribuído para que as réplicas acordem na ordem pela qual devem processar os pedidos. Este trabalho foca-se no caso em se pretende usar RME para tolerar faltas bizantinas (TFB). A solução do consenso na presença de faltas bizantinas recebeu nas últimas décadas uma atenção considerável da comunidade científica, que resultou no desenvolvimento de um conjunto significativo de protocolos distintos (e.g. [5], [8], [11], [21]), cada um otimizado para condições de execução particulares.

Neste trabalho estamos interessados no estudo de mecanismos que permitam adaptar, ou substituir, em tempo de execução um protocolo de consenso por outro mais adequado às condições de operação correntes da RME. Isto é relevante pois a maioria das aplicações práticas de RME estão sujeitas a variações do seu ambiente de execução, desde alterações na carga imposta pelos clientes a alterações no estado da rede que interliga os várias réplicas. Não existindo um

protocolo que supere todos os outros para um leque de condições de operação alargado, a única forma de assegurar um bom desempenho do sistema consiste em realizar a sua adaptação dinâmica.

O problema da adaptação dinâmica de protocolos de consenso não é novo (e.g. [9], [16]), mas a literatura é escassa para o caso bizantino e vários dos mecanismos propostos necessitam de ser modificados para operarem neste cenário. Mesmo entre aqueles desenvolvidos para contextos bizantinos não existe, tanto quanto sabemos, nenhum trabalho que permita comparar o seu desempenho. Desta forma, quem procure suportar a adaptação dinâmica tolerante a faltas bizantinas, não tem ao seu dispor dados concretos para poder escolher a técnica de adaptação que melhor se adapta às características e objetivos do sistema alvo.

Este trabalho tem, assim, dois objetivos principais. Em primeiro lugar, estuda como as diferentes técnicas de adaptação dinâmica que foram propostas para o modelo de falta por paragem podem ser aplicadas na presença de faltas bizantinas. Em segundo lugar foram concretizadas algumas destas técnicas num sistema comum, o BFT-SMaRt [2], por forma a ser possível compará-las experimentalmente. Neste contexto, o artigo apresenta um estudo comparativo do desempenho desta técnicas

2 Trabalho Relacionado

Entre os sistemas que foram propostos para concretizar replicação de máquina de estados TFB destacamos o PBFT [5], o Aardvark [8] e o Zyzzyva [11]. Cada um destes sistemas opera melhor em certas condições, sendo pior noutras, não havendo nenhum que supere todos os outros em todas as situações, como demonstrado por Singh et. al. [20]. O Zyzzyva tem melhor desempenho quando não ocorrem faltas e a rede é estável. Por outro lado, quando ocorrem faltas frequentemente o Aardvark opera melhor que os restantes, sacrificando o desempenho no caso livre de faltas. Para além disto, o desempenho do PBFT é menos sensível ao aumento do tamanho da mensagens trocadas, quando comparado com o Zyzzyva. Estas diferenças motivam o interesse de comutar entre diferentes protocolos, ou adaptar o algoritmo em execução, tirando partido das características de cada um.

Podemos dividir os sistemas que suportam a comutação de protocolos em três grandes categorias, a saber: i) sistemas que suportam a reconfiguração de protocolos monolíticos cientes da adaptação; ii) sistemas que permitem comutar entre protocolos com suporte explicito para desativação; iii) sistemas que permitem comutar entre dois protocolos arbitrários. Nos parágrafos seguintes discutimos estas diferentes aproximações ao problema da adaptação dinâmica de protocolos de consenso.

Os sistemas que suportam a reconfiguração de protocolos monolíticos cientes da adaptação assumem a existência de um único protocolo que já incorpora topos os comportamentos que se pretendem ativar em tempo de execução. Neste caso, a adaptação dinâmica consiste simplesmente em modificar um ou mais parâmetros de configuração do protocolo. Apesar deste ser o caso mais simples,

não é completamente trivial. Uma vez que os protocolos de consenso envolvem múltiplos processos, é necessário coordenar a reconfiguração das diferentes réplicas, para assegurar que estes operam em configurações compatíveis (tipicamente, com a mesma configuração). A técnica mais simples para conseguir esta coordenação, sugerida em [13], consiste em usar o próprio protocolo de consenso para definir o instante lógico em que a reconfiguração tem efeito. Para isso, a máquina de estados replicada deve suportar comandos de reconfiguração, que não são entregues à aplicação, mas que aplicam as alterações aos parâmetros, de forma a que estas alterações sejam totalmente ordenadas em relação ao fluxo das mensagem de dados. Por exemplo, o ByTAM [18], um sistema anterior que deu os primeiros passos no caminho de produzir uma versão reconfigurável do BFT-SMaRt, usa esta técnica. Uma limitação desta aproximação é que obriga a que todos os comportamentos sejam suportados por um único protocolo, de enorme complexidade e cuja correção é difícil de assegurar.

Uma solução mais modular, consiste em concretizar os diferentes comportamentos usando protocolos independentes e depois ter mecanismos para comutar entre protocolos distintos. Designamos o módulo que permite comutar entre dois ou mais protocolos um *comutador*. A comutação pode ser facilitada se os protocolos exportarem uma interface que permita ao comutador desativar o protocolo, colocando-o num estado quiescente; na literatura, estes protocolos são designados por desativáveis (do Inglês, "*stoppable*" [12]). Neste caso, após a desativação ter sido solicitada pelo comutador, as várias réplicas do protocolo coordenam-se entre si para assegurar que novas mensagens já não serão processadas. Desta forma a comutação entre dois protocolos A e B pode ser executada da seguinte forma: antes da reconfiguração o comutador envia mensagens para o protocolo A; quando a reconfiguração se inicia, o comutador pára de submeter mensagem e solicita a desativação do protocolo A: o comutador interrompe então o fluxo de mensagens, enquanto espera pela garantia que o protocolo A já está num estado quiescente; depois de obter a confirmação de desativação do protocolo A, o comutador retoma o envio de mensagem, agora através do protocolo B. Note-se que pode existir um hiato na comunicação durante a reconfiguração, uma vez que desativação de um protocolo não é instantânea e exige coordenação entre as diversas réplicas. Para além disso, nem todas as concretizações disponíveis de protocolos de consenso bizantino possuem suporte para a sua desativação, o que limita a cobertura desta aproximação.

Aublin et. al. apresentaram uma extensão a este conceito, propondo algoritmos com suporte explicito para desativação [1]. Este tipo de algoritmos permite parar a sua execução quando uma determinada condição acontece, não estando limitados a parar devido à receção de um comando. Usando estes algoritmos, os autores propõem o Abstract, uma moldura de desenvolvimento de sistemas TFB que permite desativar um algoritmo e substituí-lo por outro quando as condições de ambiente mudam. Quando um algoritmo é desativado, passa a responder a todos os pedidos com uma prova do seu término, a sua história de operações e uma indicação do próximo algoritmo a ser ativado. O cliente é então responsável por reencaminhar o seu pedido, em conjunto com os restantes dados para

o novo algoritmo a ser usado. Esta abordagem contínua a trazer um hiato na comunicação, devido à necessidade de retransmissão de pedidos. Não usando um *comutador* dedicado, necessita também de transferir mais dados pela rede (a história), a fim de garantir a correção do sistema como um todo.

Finalmente, é possível comutar entre protocolos distintos que são modelados como "caixas pretas" e que não oferecem nenhum suporte para a adaptação. A dificuldade deste cenário é que não existe uma maneira direta de um comutador identificar se um dado protocolo já se encontra quiescente, pelo que os comutadores nas diferentes réplicas têm que se coordenar explicitamente. Uma maneira de fazer esta coordenação consiste em usar um dos protocolos para enviar mensagens de controlo (designadas por *marcadores*) que são desta forma ordenadas de forma total em relação ao fluxo das mensagens de dados. Em particular, para iniciar a comutação de A para B, cada comutador pode enviar um marcador através do protocolo A; quando o primeiro marcador for entregue, todos os comutadores passam a entregar apenas as mensagens recebidas através do protocolo B. Infelizmente, devido à assincronia do sistema, um comutador pode ver várias das mensagens enviadas por si para o protocolo A ordenadas depois dos marcadores, sendo neste caso obrigado a re-enviar essas mensagens para o protocolo B. Desta forma, pode existir não só um hiato durante a reconfiguração, como um também aumento no uso da rede devido à necessidade de re-enviar mensagens de dados.

Uma estratégia para mitigar o hiato na comunicação que a reconfiguração pode introduzir consiste em, durante a reconfiguração, enviar todas as mensagens em paralelo para o protocolo A e B. O método para definir o momento lógico em que se dá a comutação pode continuar a ser o descrito anteriormente. No entanto, quando a comutação é feita, as mensagens não ordenadas por A já estarão ordenadas por B e prontas a entregar. Uma variante desta estratégia foi sugerida por [16] para protocolos tolerantes a faltas por paragem em sistemas com um detetor de faltas perfeito. Uma variante da mesma para sistemas assíncronos é discutida em [3]. Como é óbvio, uma desvantagem desta estratégia é que acarreta um aumento significativo da utilização da largura de banda durante a reconfiguração uma vez que todas as mensagens são ordenadas pelos dois protocolos, pelo que só pode ser aplicada em sistemas onde a rede não constitua um ponto de estrangulamento.

É de notar que uma técnica que utilize protocolos cientes da adaptação permite, com maior facilidade, introduzir ajustes mais finos ao modo de execução do sistema. Isto é, por exemplo, se se desejar trocar o líder do algoritmo, é possível fazê-lo alterando apenas um parâmetro da configuração. Por outro lado, em soluções recorrendo a protocolos das categorias desativáveis e caixa preta é necessário criar uma nova instância do protocolo com uma configuração inicial diferente. Para além disto, em muitos casos estes parâmetros nem são configuráveis de todo, não sendo alteráveis sem modificar o código dos algoritmos.

4

3 Adaptação de Protocolos em Contexto Bizantino

Nesta secção discutimos a concretização de diversos comutadores de protocolos na presença de faltas bizantinas. Quando os algoritmos de comutação não foram originalmente propostos para este modelo, apresentamos quais as alterações que foram necessárias aplicar aos mesmos. Apresentamos também algumas novas otimizações não discutidas na literatura.

3.1 Modelo do Sistema

Assumimos o modelo de faltas bizantinas, em que os processos que falham podem apresentar um comportamento arbitrário, incluindo entrar em conluio para atacar o sistema de forma coordenada. Não obstante, consideramos um adversário com recursos computacionais limitados, e que não consegue quebrar as primitivas criptográficas usadas pelos protocolos. Por fim, assume-se uma rede parcialmente síncrona, em que podem existir períodos de assincronia arbitrários, mas que alguma vez existe um período de sincronia em que o sistema pode fazer progresso; nos momentos de sincronia, as mensagens transmitidas são entregues ao destinatário dentro de um intervalo de tempo conhecido.

Assumimos que no sistema existem N réplicas com a capacidade de instanciar diversos protocolos de acordo bizantino. Os clientes do serviço replicado não interagem com estas instâncias diretamente, mas sim através de um componente designado por *comutador*, responsável por encaminhar estes pedidos para um ou mais destes protocolos (tornando a adaptação dinâmica transparente para os clientes). Assumimos também que existe um componente externo ao sistema, denominado o *gestor de adaptação*, que decide qual o protocolo que vai ser usado em cada momento. Quando é necessário comutar entre protocolos, o gestor de adaptação envia uma ordem a todos os comutadores, de forma a iniciar o processo de adaptação. A concretização do gestor de adaptação é ortogonal a este trabalho, estando descrita em [17]; tipicamente o próprio gestor é também replicado e cada comutador só inicia a comutação quando receber uma ordem idêntica de um quórum destas réplicas.

3.2 Adaptação Usando Algoritmos Reconfiguráveis

A adaptação usando algoritmos reconfiguráveis, apresentada por Lamport et. al.[13], é genérica e pode ser aplicada num contexto bizantino sem grandes alterações [2]. A correção da comutação depende exclusivamente das propriedades do protocolo de consenso subjacente, pelo que reconfigurarmos um protocolo tolerante a faltas bizantinas, a própria reconfiguração também tolera este tipo de faltas. Como foi referido anteriormente, esta técnica é potencialmente a mais eficiente, tanto mais que o protocolo pode ser concretizado de forma a dar prioridade aos pedidos de reconfiguração, fazendo com que estes pedidos sejam ordenados antes de outras mensagens que estejam em fila de espera para ser ordenadas.

3.3 Adaptação Usando Algoritmos Desativáveis

Num sistema tolerante a faltas por paragem, uma maneira simples de efetuar a troca entre dois protocolos desativáveis, A e B respetivamente, consiste em executar o seguinte algoritmo: cada comutador solicita a desativação do protocolo A e inibe-se de enviar mais mensagem até receber a confirmação de que A está quiescente, momento em que reinicia o envio de mensagem pelo protocolo B. No caso bizantino, este algoritmo apresenta o seguinte problema, que advém da maioria dos protocolos de consenso bizantino se basearem na eleição de um processo líder: considere-se o caso em o processo p_i é o líder do protocolo B mas é o último a ser informado que A se encontra no estado quiescente. Os restantes processos, tendo já comutado para o protocolo B podem, erradamente, assumir que ausência de atividade do líder se deve a uma falta deste (quando, de facto, p_i está correto mas bloqueado à espera da desativação de A). Isto pode dar início a processos de mudança de líder no protocolo B mesmo antes da sua operação ser iniciada em pleno. Por esta razão, num sistema tolerante a faltas bizantinas, pode ser preferível que um comutador passe de imediato a enviar mensagens no protocolo B, mesmo antes de ter a certeza que o protocolo A está quiescente. Note-se que as mensagens entregues por B terão que ser colocadas em quarentena até que A fique quiescente, para evitar entregar à aplicação a mesma mensagem mais que uma vez (as mensagens que entretanto forem ordenadas por A, devem ser descartadas da lista de mensagens ordenadas por B).

3.4 Adaptação Usando Algoritmos Não Adaptáveis

Quando é necessário comutar entre algoritmo do tipo "caixa-preta", que não fornecem nenhum suporte para a adaptação, é necessário enviar um marcador (uma mensagem de controlo dedicada ou uma *flag* associada a uma mensagem comum) no protocolo A para decidir qual o instante lógico, no fluxo de mensagens, em que se comuta para o protocolo B. Uma dificuldade adicional na presença de faltas bizantinas, consiste em verificar que o marcador corresponde a uma reconfiguração que foi de facto solicitada pelo gestor de adaptação, para evitar que um comutador bizantino possa induzir comutações indesejáveis. Considerámos duas técnicas para obter esta garantia. A primeira consiste em incluir no marcador uma prova de veracidade do pedido de configuração; esta prova consiste do comando de reconfiguração assinado por um quórum de gestores de adaptação. Outra alternativa seria iniciar a comutação apenas após receber $f + 1$ marcadores de diferentes comutadores. Esta última opção dispensaria a necessidade de enviar a prova (uma vez que ao esperar por $f + 1$ marcadores, temos a certeza que um deles foi enviado por um processo correto), mas atrasaria o processo de comutação. Por esta razão, no nosso protótipo usamos a primeira solução.

3.5 Comutação com Paralelização

Vimos anteriormente que uma maneira de reduzir o hiato que pode ocorrer durante a comutação consiste em enviar todas as mensagens nos dois protocolos

durante o processo de reconfiguração. Os trabalhos que recorrem a esta técnica, como [16] e [3], assumem que quando o processo de comutação se inicia os protocolos A e B já estão instanciados em todas as réplicas e que nenhum processo usa estes protocolos desnecessariamente. No contexto bizantino, existe o risco de um processo bizantino começar a transmitir no protocolo B mesmo que a comutação não tenha sido solicitada pelo gestor de adaptação, o que levará a um desperdício de recursos e viabilizará ataques de negação de serviço. Para minimizar este problema, desenvolvemos o seguinte estratégia. Quando um processo envia uma primeira mensagem no protocolo B deve agregar a essa mensagem uma prova de que a comutação para B foi solicitada. Tal como anteriormente, esta prova consiste do comando de reconfiguração assinado por um quórum de gestores da replicação. Um processo que recebe pela primeira vez uma mensagem para o protocolo B ativa o mesmo se a prova for válida ou descarta a mensagem e emite uma acusação ao seu emissor, se a prova for inválida.

4 Concretização

De forma a realizar uma avaliação experimental, que suporte uma análise comparativa do desempenho das diversas técnicas referidas anteriormente, concretizamos as mesmas numa moldura de software comum, o BFT-SMaRt [2]. O BFT-SMaRt é um pacote de software aberto que permite executar um serviço de máquina de estados replicada tolerantes a faltas bizantinas. Escolhemos este pacote como moldura para concretizar os comutadores uma vez que é uma das poucas concretizações de RME TFB que mantém uma equipa de suporte, e também uma das concretizações com melhor desempenho. Infelizmente, o BFT-SMaRt oferece um único protocolo de consenso, uma adaptação algoritmo descrito em [10] para a moldura Mod-SMaRt [21]. Desta forma, para ilustrar a comutação entre protocolos distintos, desenvolvemos também um protocolo adicional para o BFT-SMaRt, em particular desenvolvemos uma concretização parcial do "Fast Byzantine Consensus" descrito em [15], que denominamos Fast-SMaRt. Para além disso, acrescentamos ao BFT-SMaRt um módulo comutador, que medeia a interação entre os clientes e os protocolos de consenso suportados. O comutador está preparado para receber comandos de reconfiguração de um gestor de adaptação replicado, cujo desenvolvimento está a ser feito por outros elementos da nossa equipa [17].

Para além do comutador e do novo protocolo de consenso já referido acima, desenvolvemos também outras extensões ao BFT-SMaRt importantes para as nossas experiências. Nomeadamente, desenvolvemos versões desativáveis dos protocolos Mod-SMaRt e Fast-SMaRt para podermos testar a comutação entre protocolos com suporte para desativação, assim como a possibilidade de reconfigurar o líder destes protocolos em tempo de execução que permite comparar o tempo entre reconfigurar um único protocolo ou comutar entre duas instâncias do mesmo protocolo com diferentes configurações. Foi também necessário prover o BFT-SMaRt com suporte para a execução paralela de múltiplos protocolos; isto foi conseguido acrescentando novos campos aos cabeçalhos das mensagens e

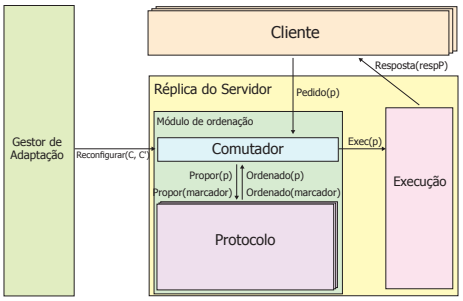


Figura 1: Arquitetura do sistema de adaptação desenvolvido.

uma camada de despacho que encaminha as mensagens para as instâncias certas. Finalmente, foi concretizado suporte para incluir nas mensagens uma prova de que a reconfiguração foi solicitada, para permitir a ativação por necessidade de novas instâncias em processo que, devido à assincronia do sistema, ainda não receberam estes comandos diretamente das réplicas do gestor de adaptação.

Note-se que seria possível otimizar a comutação tendo comutadores específicos para diferentes tipos de faltas (por exemplo, concretizando os protocolos descritos em [16] para comutar entre duas configurações tolerantes a faltas por paragem). No entanto, o protocolo atual tolera quer faltas bizantinas quer faltas por paragem, podendo ser usado para comutar entre as várias configurações possíveis.

Ao desenvolver estas extensão ao BFT-SMaRt, tivemos a preocupação de as integrar da melhor forma no atual tronco de desenvolvimento. Por exemplo, o BFT-SMaRt estava já preparado para receber alguns comandos de reconfiguração, se bem que apenas para algumas adaptações específicas como, por exemplo, a alteração em tempo de execução do conjunto de réplicas. Desenvolvemos os nossos mecanismos de suporte à adaptação, em particular o comutador, como uma extensão e generalização destes serviços. A concretização do novo protocolo seguiu também a decomposição em classes já usada na concretização do Mod-SMaRt. Uma representação da arquitetura geral do sistema é apresentada na Figura 1. Estas opções irão facilitar a manutenção do nosso código e a sua inclusão futura na distribuição do BFT-SMaRt. O código desenvolvido corresponde a aproximadamente 2k novas linhas de código (a distribuição base do BFT-SMaRt tem cerca de 28k linhas). Uma parte significativa do esforço de desenvolvimento das nossas extensões consistiu na necessidade de compreender em profundidade o código fonte do BFT-SMaRt para assegurar a integração correta das novas extensões.

5 Avaliação

Por forma a comparar as diferentes técnicas de adaptação, nesta secção pretendemos responder às seguintes questões: *a)* qual é a diferença no tempo que leva

uma reconfiguração a ser executada usando as diferentes técnicas? *b)* qual é o impacto no desempenho do sistema provocado pela reconfiguração? *c)* que carga é introduzida na rede durante o processo de reconfiguração?

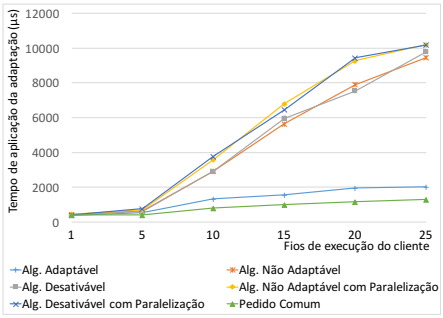
Para executar as experiências foram utilizadas 6 réplicas de BFT-SMaRt, o número mínimo exigido pelo Fast-SMaRt para tolerar uma falta bizantina [15], e um cliente capaz de introduzir carga variável no sistema. Todas as réplicas e o cliente foram hospedados em máquinas virtuais independentes no serviço DigitalOcean. Cada máquina têm dois núcleos de processamento a 2.40GHz, 2GB de memória RAM e uma ligação de rede de 1Gb/s *full-duplex*. Esta configuração foi escolhida por ser a mais poderosa, logo próxima de um servidor real, dentro dos recursos disponíveis. O gerador de carga do *micro-benchmark* envia pedidos em diferentes fios de execução, simulando múltiplos clientes. Em todas as experiências cada cliente enviou pedidos de 1kB sucessivos a cada resposta (de 10B), sem intervalo entre si (*ciclo-fechado*). O sistema foi deixado em execução durante 4 minutos antes de ser efetuada qualquer adaptação, por forma a permitir que a carga introduzida pelo cliente fizesse efeito e o sistema atingisse o seu desempenho em regime estável. Foram também feitas as experiências com 11 réplicas, tolerando duas faltas e, embora o débito do sistema tenha sido 62% inferior, o padrão observado é semelhante; estes resultados são aqui omitidos devido à falta de espaço, estando disponíveis em [4].

5.1 Tempo de Reconfiguração

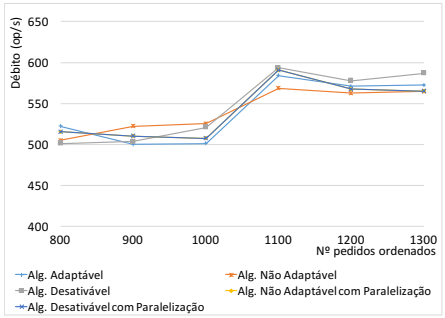
Para avaliar o tempo de reconfiguração de cada técnica foi contabilizado o tempo desde que chega um pedido de reconfiguração ao sistema até que esta é efetivamente aplicada, isto em função da carga no sistema. Os resultados são apresentados na Figura 2a. No gráfico encontra-se também representado o tempo de ordenação de um pedido comum, desde que é retirado da fila de espera até que é terminada a sua ordenação. Pode-se observar que o tempo de reconfiguração usando algoritmos adaptáveis cresce muito mais lentamente do que as restantes soluções. Mais tarde discutimos as causas para este comportamento.

5.2 Influência da Adaptação na Carga na Rede e no Débito

Para avaliar o impacto de executar uma adaptação com cada uma das soluções desenvolvidas foi usado um cliente com 20 fios de execução, tendo sido a carga que verificámos visível no sistema sem o colocar num estado de sobrecarga. Foram aferidas duas métricas: o débito de operações ordenadas por segundo e a carga na rede. O débito foi calculado a cada 100 pedidos ordenados, tendo por base o tempo decorrido desde o fim da ordenação dos 100 pedidos anteriores. Após o pedido 1000 foi submetido um pedido de reconfiguração no sistema. Os resultados obtidos estão apresentados na Figura 2b. A carga na rede foi medida durante a mesma experiência, com amostras a cada milissegundo. Por forma a facilitar a comparação entre as diferentes técnicas, foram recolhidos 40ms de amostras e estes foram alinhados temporalmente, sendo que o pedido de reconfiguração chega ao sistema aos 10ms. Os dados recolhidos estão representados

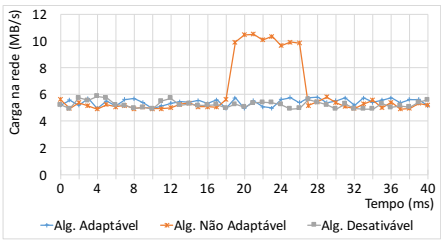


(a) Tempo de execução da adaptação

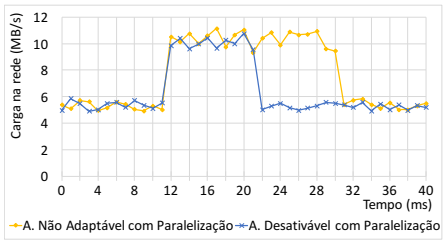


(b) Débito durante a adaptação

Figura 2: Tempo de execução da adaptação e efeito no débito



(a) Técnicas sem paralelização



(b) Técnicas com paralelização

Figura 3: Carga na rede durante um processo de adaptação

na Figura 3. Observa-se que, no contexto em que foram executados as experiências, nenhuma das técnicas utilizada induz uma penalização visível no débito dos sistema. Denota-se uma subida no débito entre os pedidos 1000 e 1100 pois a adaptação introduzida contribuiu para um aumento do desempenho do sistema. Por outro lado, a carga imposta na rede durante a adaptação varia consoante a técnica utilizada.

5.3 Análise

Nas condições em que foram executadas as experiências, num centro de dados com uma ligação de rede de baixa latência, verifica-se que qualquer uma das técnicas de adaptação consegue adaptar o sistema sem ter um impacto negativo no seu desempenho. Contudo observam-se diferenças no tempo de aplicação de uma adaptação, bem como na carga na rede introduzida pela mesma. Uma adaptação é aplicada muito mais rapidamente se recorrermos a um algoritmo adaptável, pois é possível priorizar o comando de reconfiguração em relação aos comandos de clientes. Assim, neste caso, o tempo de reconfiguração acompanha o tempo que o sistema demora a executar a ordenação de qualquer pedido. Pelo contrário, nos restantes casos, o tempo de reconfiguração depende também do

tamanho da fila de espera e da quantidade de pedidos que é necessário reenviar no novo protocolo. Esta diferença pode ser importante no desenho de um sistema se as reconfigurações tiverem por objetivo retirar o sistema de um estado de contenção ou se estas forem temporalmente sensíveis.

O uso de algoritmos não adaptáveis impõe uma sobrecarga na rede após a execução da adaptação, pois o algoritmo que deixou de estar ativo continua a executar até esgotar a fila de pedidos pendentes. Por outro lado, o uso de paralelização introduz uma carga adicional na rede antes da reconfiguração, devido à execução de ambos os algoritmos em simultâneo. Este pode ser um fator importante se o sistema operar perto do limite da largura de banda disponível, pois esta pode ser um ponto de estrangulamento do desempenho do sistema.

6 Conclusões e Trabalho Futuro

Neste artigo apresentámos e comparámos diversas técnicas de adaptação para sistemas tolerantes a faltas bizantinas. A avaliação experimental, feita num centro de dados com redes de elevado débito, mostra que neste contexto nenhuma delas penaliza significativamente o desempenho do sistema durante a adaptação. É no entanto possível observar diferenças no tempo de adaptação e na carga da rede, pelo que a escolha da técnica a utilizar pode ser relevante em sistemas que operam no limite da utilização de recursos. Apesar destas diferenças, atendendo a que a comutação para algoritmos não adaptáveis não requer qualquer alteração aos algoritmos base, no contexto da utilização em centros de dados não parece justificar-se o esforço de desenvolvimento necessário para tornar os algoritmos adaptáveis. No entanto, este comportamento poderá ser distinto em redes de maior latência. No futuro, pretendemos realizar experiências em sistemas geo-replicados e com elevada latência na rede, para comparar o desempenho dos diferentes computadores nesse contexto. O código estará disponível em github.com/cedac/calumma.

Agradecimentos Agradecemos ao Manuel Bravo e aos revisores pelos todos os comentários recebidos. Este trabalho for parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projetos com referências PTDC/ EEI-SCR/ 1741/ 2014 (Abyss), UID/CEC/ 50021/ 2013 e UID/CEC/00408/ 2013.

Referências

1. Aublin, P.L., Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 bft protocols. ACM Transactions on Computer Systems (TOCS) 32(4) (Jan 2015)
2. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with BFT-SMaRt. In: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 355–362. IEEE (Jun 2014)
3. Bortnikov, V., Chockler, G., Perelman, D., Roytman, A., Shachor, S., Shnayderman, I.: Reconfigurable state machine replication from non-reconfigurable building blocks. arXiv preprint arXiv:1512.08943 (2015)

Coerência probabilística em sistemas chave-valor escaláveis

José Ribeiro[1], Nuno Machado[1], Francisco Maia[1], Miguel Matos[2], and Rui Oliveira[1]

[1] HASLab - INESC TEC & U.Minho
[2] INESC-ID & IST, U.Lisboa

Resumo A escalabilidade é uma das características de maior relevância para os sistemas de armazenamento de dados. Para atingir uma escalabilidade e flexibilidade alta, algumas abordagens relaxam a coerência de dados evitando assim protocolos dispendiosos de coordenação. Estas abordagens são também as que tipicamente lidam melhor com churn e falhas, contudo impedem um modelo mais simples e intuitivo para o cliente, obrigando-o a preocupações como o nível de concorrência no sistema. Neste artigo apresentamos um protótipo que integra um sistema de armazenamento de dados capaz de escalar até milhares de nodos com poucas garantias de coerência, com um algoritmo probabilístico de ordem total escalável. Deste modo pretendemos que seja mitigado o problema de coerência de dados, mantendo propriedades de escalabilidade e robustez, oferecendo também ao cliente um modelo simples. Por fim, é proposta uma avaliação deste novo sistema, de maneira a verificar o impacto de garantias mais fortes de coerência no desempenho do sistema.

Keywords: Sistemas entre-pares, Protocolos de disseminação epidémicos, Rumor

1 Introdução

A utilização de modelos tradicionais cliente-servidor tornou-se obsoleta, visto ser impraticável para um sistema centralizado processar o crescente número de pedidos concorrentes das aplicações atuais [11]. Além disso, os sistemas centralizados também se caracterizam por serem pouco robustos, sendo que uma falha no servidor pode causar indisponibilidade parcial ou total no sistema. Logo, a sua utilização não é adequada para aplicações de larga escala.

De modo a resolver este problema, desenvolveram-se modelos entre-pares (do inglês *peer-to-peer (P2P)*), que conseguem operar em cenários dinâmicos e com elevados requisitos de escalabilidade [11]. Como são sistemas descentralizados e sem dependência de um ponto central, caso um nodo falhe, o serviço não é interrompido. Tipicamente, os sistemas P2P assumem que todos os nodos têm o mesmo comportamento, comunicando uns com os outros para fornecer um serviço em conjunto.

Neste artigo, focamo-nos em sistemas P2P não-estruturados baseados em protocolos de disseminação epidémica. Estes sistemas caracterizam-se pela sua simplicidade, escalabilidade e tolerância a falhas. Como são não-estruturados, não precisam de nenhum tipo de hierarquia ou estrutura, o que lhes permite lidar com altos níveis de *churn*[3] [10]. Estes protocolos têm como objetivo propagar rumores: cada nodo envia informação para um sub-conjunto de outros nodos do sistema, e cada nodo desse sub-conjunto irá por sua vez disseminar sucessivamente a informação para outro sub-conjunto de nodos [2]. Assim, a informação é propagada pela rede de forma rápida e robusta, visto que um nodo deverá receber a informação de pelo menos um dos seus vizinhos com elevada probabilidade.

Estes tipos de protocolos têm sido usados recentemente como base para desenvolver diversos sistemas distribuídos de larga escala. O DataFlasks [5, 6] é um sistema de armazenamento de dados *chave-valor* que assenta em protocolos epidémicos capaz de escalar com milhares de nodos, sendo o principal objetivo garantir disponibilidade e durabilidade dos dados. Apesar de ser altamente escalável e tolerante a faltas, o DataFlasks não garante a coerência dos dados armazenados, isto é, duas escritas concorrentes da mesma chave em nodos distintos têm a possibilidade de gerar conflitos ao guardar essa informação, podendo retornar valores diferentes em leituras dessa mesma chave. O DataFlasks assume que a gestão de concorrência, feita através de versionamento, é gerida no lado do cliente.

Tendo por base o mesmo paradigma de disseminação epidémica, o EpTO [7] foi proposto como um novo protocolo capaz de garantir probabilisticamente a entrega de mensagens com ordem total, conseguindo ainda assim escalar consideravelmente.

Neste artigo pretende-se explorar o compromisso entre coerência de dados e desempenho, através da junção do DataFlasks com o EpTO. O objetivo é poder oferecer ao cliente um nível de coerência de dados mais forte que a versão atual do DataFlasks, mantendo não só as propriedades de escalabilidade e resiliência, mas também a eficiência no acesso aos dados. Mais concretamente, pretende-se desenhar e desenvolver um protótipo de um sistema de armazenamento de dados que consiga oferecer garantias de coerência forte ao nível do tuplo tirando partido das garantias do EpTO, e simultaneamente manter a escalabilidade, disponibilidade e durabilidade do DataFlasks.

O resto deste artigo está organizado da seguinte forma. Nas secções 2 e 3 abordamos os sistemas usados para implementar o nosso protótipo. Na secção 4 mostramos a arquitetura do novo sistema, bem como cada componente utilizado. Na secção 5 são abordados os testes efetuados para validar o protótipo e os resultados correspondentes a cada teste. Por fim, as secções 6 e 7 discutem o trabalho relacionado e concluem o artigo, respetivamente.

[3] Taxa à qual os nodos entram e saem do sistema. A saída pode ser por iniciativa do nodo ou falha.

2 DataFlasks

O DataFlasks é um sistema de armazenamento de dados *chave-valor*, com objetivo de guardar informação distribuída por vários nodos para garantir a durabilidade dos dados. Como é baseado em protocolos epidémicos, as garantias de durabilidade mantém-se mesmo com níveis altos de *churn* [6]. Para comunicar com o DataFlasks, os clientes usam a seguinte interface:

- *put(K,V,D)*, que armazena a chave K com versão V e valor D no sistema.
- *get(K,V)* $\rightarrow$ *D* que obtém do sistema o valor D correspondente à chave K com versão V.
- *get(K)* $\rightarrow$ *D* que obtém do sistema o valor D correspondente à chave K com a versão mais elevada.
- *del(K,V)* que remove do sistema a chave K com versão V.

O DataFlasks assume que o controlo de concorrência é feito através do correto versionamento das chaves, feito no lado do cliente. Logo, o DataFlasks não consegue tratar operações concorrentes, caso os clientes façam escritas concorrentes da mesma chave, por exemplo *put(K1,V1,DA)* e *put(K1,V1,DB)* levando a inconsistência nos dados.

Em termos de arquitetura, cada nodo do DataFlasks é composto por 4 componentes principais. O *Request Handler* é o componente que trata os pedidos do cliente, processando-os para os outros componentes. A *Store*, como o nome diz, é o componente que guarda a informação recebida e que é responsável por persistir os dados. O *Group Construction* é o componente que divide os nodos em grupos, de modo a que cada nodo do seu grupo apenas guarde uma porção da informação total do sistema, e replique essa informação com outros nodos do mesmo grupo. E por fim o componente *Communication*, responsável por disseminar os pedidos para os restantes nodos do sistema. Mais concretamente, é composto por 3 sub-componentes. Um *Peer Sampling Service*, que periodicamente fornece ao nodo uma *view* dinâmica com nodos aleatórios do sistema. O Componente de Disseminação que aproveita os nodos da *view* do *PSS* para trocar informação com os nodos [6]. Esta propagação tem de ser confiável o suficiente para garantir que a mensagem enviada chega a alguns dos nodos do grupo. Como último sub-componente temos o mecanismo de Anti-Entropia, responsável por contactar periodicamente um nodo do mesmo grupo para trocar dados entre eles, replicando a informação e assegurando assim a disponibilidade dos dados.

Sabendo o propósito do *Group Construction*, compreende-se que os nodos não guardem toda a informação do sistema, pois isto seria custoso e pouco escalável. Assim, ao receber um pedido *put* os nodos têm a capacidade de decidir se querem guardar ou não a informação recebida de acordo com o seu grupo. De um modo geral, cada grupo apenas guarda uma fração da informação total do DataFlasks, ou seja os dados estão particionados pelos diferentes grupos. Quando um nodo decide em guardar um par chave-valor, este também envia para os nodos do respetivo grupo para replicação dos dados. E quando não consegue satisfazer um pedido, este pedido é epidemicamente propagado para os outros nodos [6].

3 EpTO

O EpTO [7] é um algoritmo de propagação epidémico de ordem total com garantias probabilísticas. Garante que cada processo concorda num conjunto de eventos recebidos com alta probabilidade, entregando-os ordenadamente para a aplicação. É completamente descentralizado e não necessita de nenhuma forma de coordenação entre processos. É altamente robusto e funciona mesmo quando os processos dependem de tempos lógicos e que não estejam sincronizados, ou quando existe *churn* ou perdas de mensagens.

A arquitetura do EpTO está implementada de maneira a que garanta ordem total nos eventos, assegurando que todos os processos entreguem todos os eventos com uma probabilidade arbitrária perto de 1. Utiliza uma adaptação do algoritmo epidémico *balls-and-bins* [3], onde os processos são abstraídos em *bins* e os eventos em *balls*. Num sistema de n processos, o processo que começa um rumor envia *balls* para K processos, escolhidos de uma maneira aleatoriamente uniforme, propagando esse rumor. Nas rondas seguintes, os processos que receberam as *balls* executam o mesmo procedimento, enviando mais *balls* para outros K processos, também escolhidos de uma maneira aleatoriamente uniforme, propagando ainda mais o rumor. O número de mensagens transmitidas por processo e por ronda é logarítmico no número de processos, e o número total de mensagens transmitidas no sistema antes de um evento ser entregue é uniforme em todos os processos.

Cada processo usa as primitivas EpTO-*broadcast*(transmissão) e EpTO-*deliver*(entrega) para comunicação, satisfazendo as seguintes propriedades:

– **Integridade**, para cada evento ***E***, cada processo EpTO-*deliver* ***E*** no máximo uma vez, se e só se ***E*** foi previamente EpTO-*broadcast*.

– **Validade**, se um processo correto[4] EpTO-*broadcast* um evento ***E***, então inevitavelmente EpTO-*deliver* ***E***.

– **Ordem Total**, se dois processos ***P*** e ***Q*** EpTO-*deliver* ambos os eventos ***E*** e ***E'***, então ***P*** EpTO-*deliver* ***E*** antes de ***E'*** se e só se ***Q*** EpTO-*deliver* ***E*** antes de ***E'***.

– **Acordo probabilístico**, se um processo EpTO-*deliver* um evento ***E***, então com alta probabilidade todos os processos corretos inevitavelmente vão EpTO-*deliver* ***E***.

O EpTO é composto por dois componentes. O Componente de Disseminação lida com a receção e propagação dos eventos e é responsável por satisfazer a propriedade de acordo probabilístico.

O Componente de Ordenação recebe os eventos do Componente de Disseminação, ordena-os e entrega-os para a aplicação, satisfazendo a propriedade de ordem total. As outras duas propriedades, integridade e validade são atingidas pelos dois componentes em conjunto. A principal tarefa deste componente é mover os eventos recebidos para um conjunto de eventos já entregues à aplicação para prevenir eventos entregues duplicados, preservando ao mesmo tempo a ordem total dos eventos. À exceção da receção de pedidos, o Componente de

[4] processo que não falha

Disseminação é executado em rondas de δ unidades de tempo, e no fim de cada ronda executa o Componente de Ordenação.

4 Desenho e Implementação

Com a integração do EpTO no DataFlasks pretendemos resolver os problemas de inconsistência previamente descritos. O objetivo é obter um sistema que traz o melhor de dois mundos, a escalabilidade e robustez do DataFlasks, com a escalabilidade e garantias de coerência do EpTO. Esta garantia de coerência é somente probabilística, pois não existem certezas determinísticas que todos os nodos do sistema recebem os pedidos pela mesma ordem. Daí a necessidade de propagar um número logarítmico de mensagens suficientes para que essa informação chegue a todos os nodos com alta probabilidade.

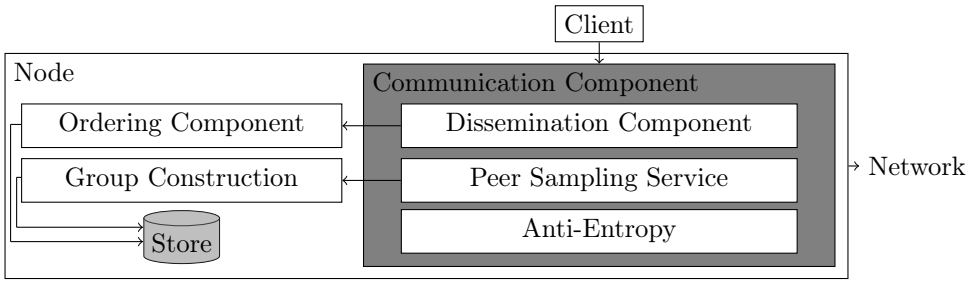


Fig. 1: Arquitetura do novo sistema

A sua arquitetura está ilustrada na Fig. 1. A interface do cliente é igual ao DataFlasks com a exceção que o cliente já não precisa de especificar a versão nas operações de *put* e *get*. Esta gestão é feita internamente pelo sistema, sendo a versão correspondente à ordem total dada pelo EpTO, logo apenas é guardada a última versão de cada chave. Isto garante que clientes concorrentes vão ter sempre uma vista coerente do sistema.

Os pedidos dos clientes são enviados para o Componente de Disseminação, que mantém o comportamento do EpTO, ou seja, após receber um pedido este é processado e de seguida adicionado à lista de *balls* para transmissão. O *Request Handler* do DataFlasks é responsável por evitar entrega de mensagens duplicadas. No entanto, uma vez que com o EpTO garantimos a integridade do sistema, não é necessário utilizar o *Request Handler* no DF+EpTO. De seguida os pedidos passam pelo Componente de Ordenação antes de chegarem à *store*, para garantir que estes são processados com ordem e que não violam as propriedades de ordem total.

Por fim, quando um pedido chega à *store*, como dito anteriormente, apenas é guardada a última versão escrita para uma dada chave[5]. Caso a chave seja

[5] Esta limitação prende-se somente com o consumo de recursos da máquina usada na Avaliação (Secção 5). Na prática o sistema poderia guardar um número parametrizável de versões.

adicionada pela primeira vez, esta é marcada com a versão 0, e futuras escritas na mesma chave incrementam essa versão. Como os pedidos passam previamente pelo Componente de Ordenação, é garantido com alta probabilidade que, após um *time-to-live* predefinido, todos os nodos vão executar todas as operações pela mesma ordem e que não existem nodos com valores diferentes para uma dada chave e versão.

Tanto o DataFlasks como o EpTO utilizam o *PSS* Cyclon, que periodicamente escolhe um nodo da sua *view* para permutar referências para outros nodos de modo a manter as propriedades da vista mesmo na presença de faltas e *churn* [10]. Como tem o mesmo comportamento em ambos os sistemas, não faz sentido incorporar dois serviços, portanto é utilizado um que é partilhado por ambos. Tal como no DataFlasks, uma cópia destas referências é enviada para o *Group Construction* para o cálculo do grupo correspondente [6]. Para que tenhamos um sistema com os grupos uniformemente distribuídos, é atribuído a cada nodo uma posição gerada de um modo uniforme entre $[0, 1]$. Assim prevenimos a possibilidade de existirem grupos com muita discrepância no tamanho, que vai ajudar na avaliação do sistema com casos de *churn*.

Por último temos o mecanismo de Anti-Entropia, que como no DataFlasks serve para replicar informação da *store* entre nodos do mesmo grupo, para garantir a disponibilidade dos dados. Numa primeira abordagem, apenas usamos este mecanismo para trocar chaves que os nodos estivessem a replicar. Contudo, como a *store* está dependente de versões de chave-valor, decidimos otimizar este mecanismo ao trocar também as chaves-valor com a versão mais recente quando comparadas entre os nodos.

5 Avaliação

Para avaliar este novo sistema, usamos o mesmo simulador que foi utilizado no EpTO [7]. Este simula um sistema distribuído e permite modelar assincronia de rede e processos, *churn*, perda de mensagens e latências e assimetria na rede. O simulador está implementado na linguagem *Python*, bem como os dois protótipos desenvolvidos neste trabalho. Para aferir corretamente as vantagens e desvantagens do sistema proposto, foi também implementado o DataFlasks original. Foram avaliados vários critérios, quer de correção, quer de desempenho, que são descritos nas secções seguintes. Por omissão, os resultados apresentados correspondem à média de 5 execuções independentes.

5.1 Configurações

O *workload* base contém um sistema com 300 nodos executado durante 500 ciclos. Tal como no EpTO, a latência entre nodos foi parametrizada tendo como base uma amostra de nodos do *PlanetLab* geograficamente dispersos [7]. Para configurações específicas dos protocolos usados, o DataFlasks usa uma *view* Cyclon com tamanho $ln(300) \approx 6$. Antes de cada simulação, cada nodo é inicializado com um conjunto de nodos aleatórios do sistema [6]. Para o tamanho

aceitável a *view* do *Group Construction* é entre $[7, 13]$, e para esta configuração, o número expetável de grupos é 32.

De um modo geral, para cada teste, o sistema é executado até ficar estável (ou seja, até os grupos DataFlasks estarem bem formados) e populado com um conjunto inicial de valores. Depois desta fase inicial são executadas operações dos clientes consoante o *workload* específico e tiradas as métricas relevantes.

Foram executados os seguintes testes: débito, latência, número de mensagens trocadas entre nodos, que permite aferir o custo e desempenho do sistema. Medimos também a coerência do sistema e a durabilidade dos dados de modo a garantir que as propriedades que o DataFlasks oferece não se perderam ao integrar com o EpTO.

Cada teste foi executado em 3 situações diferentes, em que duas delas são o DataFlasks isolado com diferentes configurações, e como última o DataFlasks com o EpTO. Simulamos estas 3 situações para mostrar as diferenças presentes neste novo sistema. Para futura referência, chamamos a estes sistemas **DF 1/3**, **DF-All** e **DF+EpTO**, em que:

1. **DF 1/3**, o cliente só pode efetuar outro pedido quando receber um terço da média dos limites do intervalo do *Group Construction* (neste caso $(13+7)/2/3 = 3$ respostas); 2. **DF-All**, o cliente precisa de receber o limite inferior do intervalo (neste caso 7 respostas); 3. **DF+EpTO**, o cliente apenas precisa de receber uma resposta para iniciar outro pedido.

Os dois primeiros sistemas estão configurados desta maneira uma vez que se esperarmos só por uma resposta, não temos a garantia que existe um nível de replicação suficiente. No **DF+EpTO** sabemos que todos os nodos irão receber os pedidos pois temos garantias mais fortes de entrega de mensagens. Para cada teste, submetemos os 3 sistemas a 0%, 1%, 5% e 10% de *churn* de 5 em 5 ciclos. Isto é executado removendo a percentagem, previamente configurada, de nodos vivos do sistema e logo de seguida é inserido no sistema a mesma percentagem de nodos. Estes nodos também recebem uma *view* inicial de nodos aleatórios vivos do sistema.

Para simular os sistemas descritos executamos 10 clientes que fazem operações de leitura (*get*), inserção (*put* de uma chave nova) e atualização (*put* de uma chave existente) com 80%, 15% e 5% de probabilidade respetivamente. Escolhemos este *workload* em geral pois num ambiente realista, os clientes fazem mais operações de leitura do que escrita ou atualização. Para tornar os testes mais consistentes, cada cliente tem sempre ao seu dispor uma lista dos nodos vivos do sistema.

Exemplificando o comportamento de um cliente: 1. Para os testes com DataFlasks, as operações mantêm o comportamento deste sistema, em que as escritas recebem uma chave, uma versão e um objeto como *input*, e as leituras recebem uma chave e versão. Para os testes no novo sistema, o comportamento é igual ao do EpTO, em que as operações só recebem chave e valor, ou só uma chave respetivamente; 2. O cliente faz uma operação e guarda a chave e versão pedida (em testes de DataFlasks), ou apenas a chave (em testes no novo sistema). Adicionalmente também guarda o *timestamp* atual; 3. Quando recebe uma resposta correspondente ao pedido guardado, incrementa o contador de respostas recebidas.

Quando tiver o número de respostas supostas configuradas previamente, guarda o pedido numa lista, calcula o tempo que demorou até receber a última resposta e guarda esse valor. Depois disto, é feito um novo pedido seguindo a distribuição acima apresentada. Ou seja, os clientes não utilizam *think-time* e submetem novas operações imediatamente.

Como configuração adicional, cada cliente envia cada pedido para um número parametrizável de nodos aleatórios. Assim conseguimos tolerar o impacto do *churn* no ponto de vista do cliente que, nesta implementação, não tenta executar outra vez o mesmo pedido em caso de falha. Este parâmetro é 3, quer nos cenários com *churn*, quer nos cenários sem *churn* o que nos permite uma comparação mais justa entre ambos os cenários.

5.2 Resultados

Coerência Para este teste apenas foi preciso uma simulação, para mostrar que o DataFlasks tem fraca coerência de dados. Para obter estes dados, foram guardadas as chaves e valores existentes no sistema de 50 em 50 ciclos.

Foi adicionada uma configuração em dois clientes, em que a primeira operação é uma escrita com a chave *HelloWorld* e valor aleatório. Se após as escritas dos clientes terminarem tivermos vários valores para a mesma chave, estamos perante um problema de coerência no sistema.

A Fig. 2 foi obtida num sistema sem *churn*, e cada escala de cinzento presente no gráfico simboliza um valor diferente para a chave *HelloWorld*. Ao analisar o mesmo, podemos ver que num grupo com 9 nodos, a chave *HelloWorld* tem dois valores diferentes (correspondente à inserção de cada um dos clientes) quer no **DF 1/3** quer no **DF-All**. No **DF+EpTO** podemos provar que o sistema mantém a mesma chave-valor em todo o grupo, preservando assim a coerência dos dados.

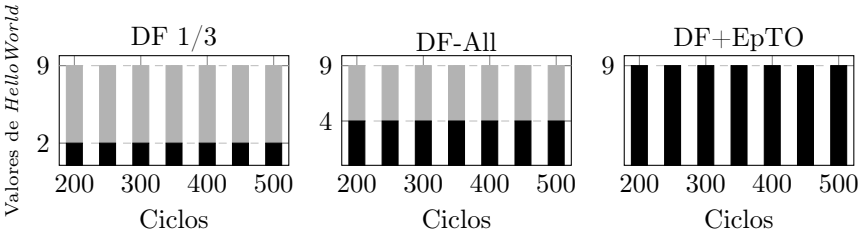


Fig. 2: Coerência nos diversos tipos de sistema com 0% de *churn*

Durabilidade Neste teste queremos mostrar que o **DF+EpTO** não destrói as propriedades que o DataFlasks sozinho oferece. Portanto, foi calculado para as várias percentagens de *churn*, o número de chaves distintas que estão guardadas

no sistema antes, durante e após a ocorrência de *churn*. Neste teste, só se executa a fase inicial de popular o sistema. Depois disto, o *churn* é ativado. Não foi testada esta propriedade com os clientes ativos porque com a contínua inserção de novas chaves tornaria-se difícil de mostrar a durabilidade do sistema. Pela Fig. 3 mostramos que esta propriedade mantém-se, e que mesmo com um *churn* de 10%, o número de chaves distintas mantém-se, garantindo assim a durabilidade.

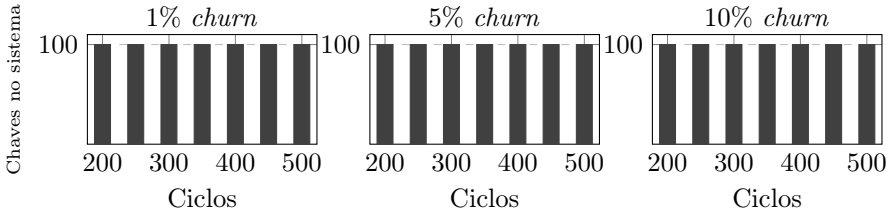


Fig. 3: Durabilidade no novo sistema com diferentes percentagens de *churn*

Latência Os resultados deste teste foram obtidos agregando as listas das latências de cada cliente dos 5 testes executados, e depois disso, calculando o *CDF* para cada sistema nas diversas percentagens de *churn*. Através da Fig. 4, podemos ver que em ambos os sistemas de DataFlasks, a latência tem valores muito baixos comparativamente ao **DF+EpTO**. Isto deve-se ao facto de com o EpTO, cada pedido precisa de ser processado de modo a que seja entregue ao cliente de uma forma ordenada [7], demorando muito mais tempo. Como o DataFlasks não tem um comportamento tão exigente como o EpTO, os pedidos são entregues ao cliente de uma forma direta.

Para os 2 sistemas de DataFlasks a 10% de *churn*, a latência assemelha-se à latência do **DF+EpTO**. Isto ocorre devido à necessidade de o cliente com estes dois sistemas precisar de receber mais que uma resposta, e como os sistemas estão sob *churn* constante, é expectável que os nodos mais recentes demorem a receber o pedido para responder ao cliente. Daí existir uma diferença maior de *timestamps*.

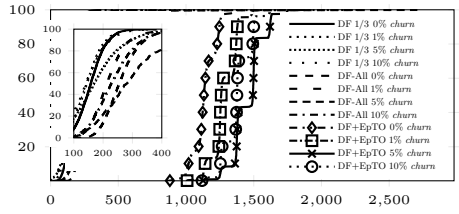


Fig. 4: CDF da latência para os diversos tipos de sistema e % de *churn*. O eixo dos *xx* indica os diferentes instantes da simulação.

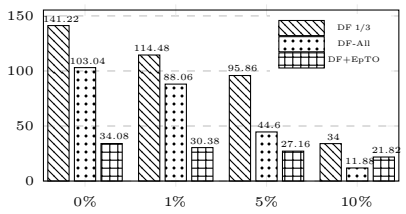


Fig. 5: Débito para os diversos tipos de sistema e % de *churn*.

Débito Para testar o débito de cada sistema, calculamos o tamanho da lista de pedidos de cada cliente nos 5 testes, e determinamos a média resultante. Na Fig. 5 mostramos a média dos pedidos que os clientes fizeram nos 3 sistemas com diferentes níveis de *churn*. Daqui podemos retirar que o **DF 1/3** consegue processar mais pedidos de clientes em qualquer percentagem de *churn*, pois os clientes apenas precisam de receber 3 respostas, sendo capazes de fazer mais pedidos. Com o **DF+EpTO**, apesar de o cliente precisar de apenas receber uma resposta, como o processamento das mensagens é mais custoso, a latência é mais alta que faz com que o débito seja o menor dos 3 sistemas testados. Para o caso do **DF-All** com 10% de *churn*, o débito é mais baixo que o **DF+EpTO** devido ao número de respostas configurado anteriormente. Como é um número grande para um sistema de 300 nodos sob 10% de *churn*, a possibilidade de existirem nodos suficientes para responder ao cliente é baixa, que faz com que o cliente fique à espera mais tempo, que por consequência faz menos pedidos.

Número de mensagens Com este teste queremos mostrar o número de mensagens que os sistema troca entre nodos durante a simulação. Neste número estão incluídos as trocas periódicas de informação do Cyclon, do mecanismo de anti-entropia, e a disseminação das operações dos clientes para o sistema todo.

Analisando a Tabela 1, podemos ver que o **DF+EpTO** tem um custo muito mais alto. Em ambos os sistemas de DataFlasks, e DataFlasks com EpTO, o Cyclon e o mecanismo de anti-entropia são independentes, portanto a diferença de número de mensagens aparece na maneira de como cada sistema propaga as operações dos clientes. O DataFlasks é menos custoso porque propaga os pedidos dos clientes de uma forma ativa, isto é, ao receber o pedido, processa logo a mensagem e dissemina para os nodos da sua *view*. Ao contrário do **DF+EpTO**, que apesar de executar periodicamente o Componente de Disseminação, o custo de uma mensagem é maior, pois com EpTO a mensagem tem o custo de uma *ball*, ou seja, uma lista de pedidos.

Tabela 1: Número de mensagens enviadas nos diversos tipos de sistema e *churn*

Sistema \ ***Churn***	0%	1%	5%	10%
DF 1/3	3 233 681	2 581 068	2 155 488	1 286 147
DF-All	2 569 204	2 150 162	1 373 956	868 917
DF+EpTO	9 627 482	7 687 878	6 312 383	4 870 851

6 Trabalho Relacionado

Como uma das bases mais importante dos sistemas abordados ao longo deste artigo, temos o protocolo Cyclon [10], um *Peer Sampling Service* baseado em protocolos epidémicos. O objetivo do Cyclon é manter uma rede virtual sobreposta

(*overlay*). Cada nodo comunica periodicamente com outros nodos do sistema escolhidos aleatóriamente de modo a garantir que a rede permanece conectada mesmo com elevados níveis de *churn*. Ao comunicar e ao manter informação sobre, um número limitado de nodos, o Cyclon consegue escalar logaritmicamente para sistemas com um elevado número de nodos. Como o Cyclon tem como objetivo a disseminar a informação através de uma *view* aleatória, a sua construção é baseada em sistemas *peer-to-peer* não-estruturados, pois sistemas estruturados necessitam de manter uma estrutura específica para garantir leituras eficientes à custa de robustez.

Da mesma forma, sistemas de armazenamento de dados baseados em sistemas estruturados, como não lidam com altos níveis de *churn*, não são práticos para cenários de larga escala. Temos como exemplo o Dynamo [1] e o Cassandra [4], que utilizam uma *Distributed Hash Table* para armazenar os dados. Contudo, ao relaxar a sua consistência de dados conseguem de escalar com milhares de nodos. DataFlasks tem como objetivo manter um nível de escalonamento alto mesmo em presença de *churn*, utilizando um sistema *peer-to-peer* não-estruturado.

Como o problema de consistência ainda persiste em muitos sistemas, algoritmos de ordem total têm sido estudados para resolver este problema. Algumas abordagens focam-se em garantias determinísticas, garantias que são difíceis de alcançar devido ao seu custo elevado e à incapacidade de escalar. Através de protocolos epidémicos construiu-se algoritmos probabilísticos como o EpTO, criado para garantir as propriedades de ordem total sem depender de sincronismo entre processos ou mesmo com latência alta. Ao ter como base os protocolos epidémicos, também oferece garantias de escalabilidade e robustez mesmo com altos níveis de *churn* e perda de mensagens [7]. Outras abordagens baseiam-se no uso de relógios vetoriais [8] ou tipos replicados livres de conflito [9].

Este protótipo foi proposto para mostrar a possibilidade de um sistema de armazenamento de dados capaz de manter as propriedades de escalabilidade resolvendo ao mesmo tempo o problema de coerência de dados.

7 Conclusão e Trabalho Futuro

Neste artigo apresentamos um sistema P2P descentralizado não-estruturado que combina o DataFlasks com o EpTO, de forma a obter um novo sistema capaz de resolver os problemas de coerência de dados do DataFlasks, garantindo simultaneamente a alta escalabilidade e robustez do sistema, e a disponibilidade e durabilidade dos dados.

Através da análise efetuada, podemos concluir que a integração do EpTO com o DataFlasks continua a oferecer a disponibilidade e durabilidade dos dados, reforçando as garantias de coerência de dados, à custa de um débito mais baixo, latência maior e mais recursos consumidos ao propagar os pedidos dos clientes, tornando o sistema mais sobrecarregado.

Como trabalho futuro é preciso analisar os mesmos sistemas com *workloads* mais apropriados e mais realistas, como por exemplo, um *workload* com milhares de nodos e clientes. Adicionalmente, podemos fazer melhorias na parte